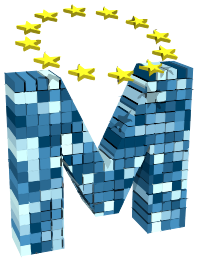


Project title: Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
Project acronym: MOSAICrOWN
Funding scheme: H2020-ICT-2018-2
Topic: ICT-13-2018-2019
Project duration: January 2019 – December 2021

D5.5

Report on Data Sanitisation and Computation

Editors: Jonas Böhler (SAP SE)
Reviewers: Merry Globin (EISI)
Pierre-Antoine Champin (W3C)

Abstract

This document reports on the innovation with regards to sanitization and collaborative computation developed during MOSAICrOWN. The collaborative scenario considers multiple parties that perform a joint computation and the presented techniques support a joint anonymization process and privacy-preserving statistical analysis.

First, the document reports on advancements with regards towards scalable anonymization in a distributed setting, where parties jointly anonymize a large dataset which can be distributed and might not fit entirely in main memory due to its size. The basis for the distributed algorithm is Mondrian, an efficient anonymization algorithm for a centralizied setting, which is extended to support multiple worker processes. Building on D5.4, further advancements and empirical evaluations are detailed in the first chapter of this document that show the scalability of the presented approach.

Then, the document reports on secure computations of differentially private statistics with a focus on the multi-party setting (i.e., more than two parties). The solution presents an efficient cryptographic protocol that does not reveal the input of any party and only provides an anonymized output. The presented protocol supports decomposable aggregate functions (based on, e.g., ranks, counts, loss as in empirical risk minimization) as used in MapReduce-style frameworks and is extensively evaluated.

Type	Identifier	Dissemination	Date
Deliverable	D5.5	Public	2021.10.31

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825333.

MOSAICrOWN Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	EMC Information Systems International	EISI	Ireland
3.	Mastercard Europe	MC	Belgium
4.	SAP SE	SAP SE	Germany
5.	Università degli Studi di Bergamo	UNIBG	Italy
6.	GEIE ERCIM (Host of the W3C)	W3C	France

Versions

Version	Date	Description
0.1	2021.10.05	Initial Release
0.2	2021.10.26	Second Release
1.0	2021.10.31	Final Release

List of Contributors

This document contains contributions from different MOSAICrOWN partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	Jonas Böhler (SAP SE)
Chapter 1: Scalable Distributed Data Anonymization	Sabrina De Capitani di Vimercati (UNIMI), Dario Facchinetti (UNIBG), Sara Foresti (UNIMI), Giovanni Livraga (UNIMI), Gianluca Oldani (UNIBG), Stefano Paraboschi (UNIMI), Matthew Rossi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 2: Secure Multi-Party Computation of Differentially Private Statistics	Jonas Böhler (SAP SE)
Chapter 3: Conclusions	Jonas Böhler (SAP SE)

Contents

List of Figures

List of Tables

Executive Summary

This document presents two solutions for sanitization and computation in a collaborative manner developed during WP5 of MOSAICrOWN.

The generation and collection of data continuously increases as computation devices become ubiquitous (e.g., smartphones, IoT sensors). Such data contains sensitive information – such as personal data or business-critical process data – and requires special care and protection in the form of sanitization. Furthermore, as the amount of data increases it becomes harder to scale the sanitization efforts and connect the data hidden in distributed data silos.

This document tackles these challenges, namely, scalability and distributed processing.

Chapter 1 details the advancements and progress for scalable anonymization based on Mondrian which satisfies k-anonymity and l-diversity. While Mondrian initially considered only a centralized setting, the solution presented here extends the technique to a distributed setting. Multiple parties can collaboratively anonymize their data via dynamic partitioning that enables parallelized processing. Furthermore, various strategies for the generalization performed during the sanitization are supported. The approach is empirically evaluated and shown to provide efficient scalability.

Chapter 2 describes a secure multi-party computation for differentially private statistics. In more detail, the solution allows multiple parties holding data that cannot be easily shared to securely compute an anonymized statistic that satisfies differential privacy without learning any data from the other parties. Various statistics are supported that can be expressed via decomposable aggregate functions (e.g., counts, ranks) as found in frameworks based on MapReduce. The presented solution is based on probabilistic selection via the exponential mechanism where each possible output element for a target statistic is selected with a probability proportional to its utility, e.g., closeness to the desired output. To ensure efficiency, the main challenges of probabilistic selection are addressed as follows. First, the potentially very large data domain is partitioned in subranges and the subrange with the highest utility is selected. This is repeated, until the subrange is small enough, e.g., contains only the desired element. Second, the computation of the selection probability requires exponentiation, which is prohibitively expensive in secure computation, and more efficient alternatives are presented.

1. Scalable Distributed Data Anonymization

In this chapter, we present the advancements in the models and techniques developed in MOSAICrOWN for addressing the problem of efficiently anonymizing large data collections. In particular, this chapter illustrates a solution that extends Mondrian [LDR06], an efficient and effective approach originally proposed for achieving k-anonymity in a centralized scenario, to operate to enforce both k-anonymity and ℓ-diversity in a distributed scenario. With our approach, anonymization is executed in parallel by a set of workers, each operating on a portion of the original dataset to be anonymized. The design of our partitioning approach aims at limiting the need for workers to exchange data, by splitting the dataset to be anonymized into as many partitions as the number of available workers, which can independently run our revised version of Mondrian on their portion of the data. A distinctive feature of our proposal is that the partitioning approach does not require knowledge of the entire dataset to be anonymized, but can be executed on a sample of the dataset whose size can be dynamically adjusted, being therefore applicable in scenarios where the dataset is large, possibly distributed, and does not entirely fit in main memory. The advanced techniques illustrated in this chapter have been implemented by a tool, illustrated in Deliverable D5.4 [PFOR21]. In this chapter, we also provide an update on the architecture and on the experimental results illustrated in D5.4 [PFOR21].

The reminder of this chapter is organized as follows. Section 1.1 discusses the state of the art in data anonymization and the innovation produced by MOSAICrOWN. Section 1.2 illustrates basic concepts. Section 1.3 presents an overview of our approach for distributed anonymization, modeling the reference scenario and illustrating the actors involved and a sketch of our anonymization solution. Section 1.4 discusses our approach for partitioning the dataset to be anonymized in portions assigned to the workers for anonymization. Section 1.5 illustrates how workers can independently anonymize their portion of data and evaluate the information loss. Section 1.6 presents an update of the tool implemented for the distributed anonymization. Finally, Section 1.7 illustrates an update on the experimental results.

1.1 State of the Art and MOSAICrOWN Innovation

In this section, we illustrate the state of the art and the innovation produced by MOSAICrOWN for scalable distributed anonymization of data.

1.1.1 State of the Art

Effectively protecting the privacy of the respondents of a data collection is a widely investigated problem (e.g., [CDFS07, CDFS09, DFLS11], but traditional solutions based on either k-anonymity [LDR05,LDR06,Sam01] and its extensions (e.g., [DFLS12,LLV07,LLV10,MKGV07]), or differential privacy [Dwo06] and its extensions (e.g., [DR14]) typically assume a centralized setting for the enforcement of anonymization.

Some works have investigated the problem of achieving scalable runtime performance over large datasets (e.g., [ZYLC13, ZLD+16, CWR14, AKS21, SA17]). These works distribute the dataset to be anonymized to a set of workers, in charge of anonymization. However, these solutions require several data exchanges among the nodes participating in the distributed anonymization [CWR14], focus on specific machine learning scenarios [AKS21], or do not apply the Mondrian algorithm for anonymization [ZYLC13, ZLD+16, SA17].

Other works have investigated anonymization of distributed data and/or multiple datasets (e.g., [TG12, KPEK14, DYL+13, JX09]). However, these solutions focus on scenarios characterized by multiple sources [TG12, KPEK14], possibly with different privacy requirements [DYL+13] and with multiple data owners having restricted visibility on the datasets involved in the computation [JX09].

Alternative approaches for protecting data are based on data fragmentation, possibly coupled with encryption. These solutions have been proposed for enforcing ℓ-diversity [XT06] and/or to break generic sensitive associations among data (e.g., [CDF+10, DFJ+15]). These proposals produce different fragments (views) of the original data collection.

1.1.2 MOSAICrOWN Innovation

MOSAICrOWN produced several advancements over the state of the art, which are discussed in this section.

- The first innovation is the definition of an extended version of the original Mondrian approach to operate in a distributed scenario leveraging the computational power of a set of workers operating in parallel in a cluster, enforcing both the k-anonymity and ℓ-diversity requirements and limiting the need for workers to exchange data.
- The second innovation is the definition of different strategies that permit to partition large datasets, and to assign such partitions to workers for anonymization, while requiring visibility only over a sample of the dataset to be anonymized.
- The third innovation is the support for different generalization strategies, including the use of ontology-based generalization hierarchies that permit to produce semantically-aware anonymizations, and for different metrics for assessing the information loss.

Some of the results obtained by MOSAICrOWN and illustrated in this chapter have been published in [DFF+21a, DFF+21b]. The proposal illustrated in this chapter has been implemented by the tool presented in Deliverable D5.4 [PFOR21].

1.2 Basic Concepts

Our solution is based on three main pillars: k-anonymity, ℓ-diversity, and Mondrian.

***k*-Anonymity.** k-Anonymity [Sam01] is an anonymization approach that pursues a privacy requirement demanding that no release of data can be related to less than a certain number k of individuals. k-Anonymity has been designed to counteract the risk that a de-identified dataset (i.e., where identifying information such as social security numbers have been removed) be linked to other, non de-identified datasets through *quasi-identifying* attributes such as sex, date of birth, and city (denoted QI), available in both datasets. Such linking attack may in fact re-identify (some

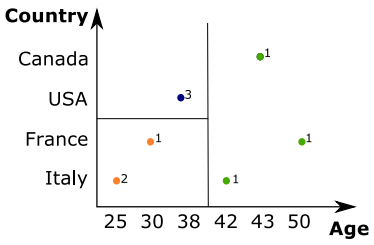
of the) de-identified respondents. To illustrate, consider a de-identified medical dataset including a record for a male patient, born on 1970/06/15 and living in a small village. If this combination of values is unique in the external world (i.e., the village only has one male citizen born on 1970/06/15), it may be sufficient to link the de-identified medical dataset with the village voters list to re-identify his record. k-Anonymity modifies the values of the attributes in the QI to ensure that no tuple can be uniquely related to its respondent through her QI values, and vice-versa. Practically, k-anonymity ensures that each combination of quasi-identifying values in a dataset appears with at least k occurrences. In this way, each individual in any external data source could be mapped to 0 or at least k records in the anonymized dataset. This guarantee can be obtained in different ways. The original proposal of k-anonymity applies data generalization to the attributes in the QI, which may be used for linking to external data sources [Sam01]. Data generalization is a data protection technique that replaces data values with other, more general values. For instance, an individual's complete date of birth ⟨`year/month/day`⟩ can be generalized to ⟨`year/month`⟩, or just to ⟨`year`⟩. Since generalization (while maintaining data truthfulness) removes details from data, it reduces the risk of finding unique correspondences with external data sources. With respect to the example above, a 3-anonymous version of the medical dataset would generalize the patients' sex, date of birth and city until each combination of values appears with at least 3 occurrences: in this way, no record in the village voters list could be mapped (through attributes in the QI) to less than 3 records in the medical dataset.

ℓ-Diversity. ℓ-Diversity [MGK06] extends k-anonymity to protect the values assumed by the attributes in the dataset that are considered sensitive. Consider a 3-anonymous medical dataset, and an equivalence class E (i.e., the set of tuples sharing the same values for the quasi-identifying attributes) where all records have value *Stroke* for sensitive attribute `Disease`. While no record in an external data source (e.g., a voters list) can be uniquely mapped to any of these tuples, the 3-anonymous dataset still leaks the fact that all individuals in the voter list that map to E suffered a stroke. ℓ-Diversity extends k-anonymity by demanding that each equivalence class contains at least ℓ well-represented values for the sensitive attribute. Several definitions of *well-represented* have been proposed, and a natural interpretation requires at least ℓ different values for the sensitive attributes.

Mondrian. Mondrian [LDR06] is a multi-dimensional algorithm that has established itself as an efficient and effective approach for achieving k-anonymity. Mondrian leverages a spatial representation of the data, mapping each quasi-identifier attribute to a dimension and each combination of values of the quasi-identifier attributes as a point in such a space. Mondrian then recursively cuts the tuples in each partition (the whole dataset at the first step) based on their values (e.g., for numerical attributes, whether lower/higher than the median) for a quasi-identifying attribute chosen at each cut. The algorithm terminates when any further cut would generate sub-partitions with less than k tuples, at which point values of the quasi-identifier attributes in a partition are substituted with their generalization. Figure 1.1(b) shows the spatial representation and partitioning of the dataset in Figure 1.1(a), where the number associated with each data point is the number of tuples with such values for the quasi-identifier (composed of attributes `Age` and `Country`) in the dataset. Figure 1.1(c) illustrates an example of a 3-anonymous version of the dataset. In this example, the dataset is first partitioned based on the values of attribute `Age`, obtaining two partitions where all tuples with `Age` less than or equal to the median value 38 are in partition p_1, and all the remaining tuples are in the other partition p_2. Of these partitions, only p_1 can be further partitioned, since p_2 contains only three tuples and any splitting among them would clearly violate

Age	Country	TopSpeed
25	Italy	132
25	Italy	132
30	France	128
42	Italy	110
50	France	115
43	Canada	115
38	USA	126
38	USA	127
38	USA	140

(a)



(b)

Age	Country	TopSpeed
[25,30]	Europe	132
[25,30]	Europe	132
[25,30]	Europe	128
[42,50]	World	110
[42,50]	World	115
[42,50]	World	115
38	USA	126
38	USA	127
38	USA	140

(c)

Figure 1.1: An example of a dataset (a), its spatial representation and partitioning (b), and a 3-anonymous and 2-diverse version (c), considering quasi-identifier `Age` and `Country` and sensitive attribute `TopSpeed`

3-anonymity. Partition p_1 is then further partitioned based on the values of attribute `Country`, obtaining two new partitions $p_{1,1}$ and $p_{1,2}$ where all tuples with `Country` equal to Canada or USA are in $p_{1,1}$, and the remaining tuples are in $p_{1,2}$. No further partition is now possible ($p_{1,1}$, $p_{1,2}$ and p_2 have all three tuples each), and hence the quasi-identifier attributes in each partition can be generalized to obtain 3-anonymity. For example, tuples in partition $p_{1,1}$, with original `Age` values equal to 25 and 30, and original `Country` values equal to Italy and France, would have their `Age` generalized to the range $[25,30]$ and `Country` generalized to Europe.

1.3 Distributed Anonymization

We consider a scenario where a large and possibly distributed dataset $\mathscr{D}$ needs to be anonymized, and may not entirely fit into the main memory of a single machine. Our goal is then to distribute the anonymization of $\mathscr{D}$ to a set $W = \{w_1, \ldots, w_n\}$ of workers so that they can operate in parallel and independently from one another, to have benefits in terms of performance while not compromising on the quality of the solution (with respect to a 'traditional' centralized anonymization of $\mathscr{D}$). To solve this problem, we have extended Mondrian to operate in such a way that workers in W are assigned non-overlapping portions of $\mathscr{D}$ (i.e., sets of tuples of $\mathscr{D}$, which we call *fragments*), and each worker $w \in W$ can independently anonymize its fragment respecting both k-anonymity and ℓ-diversity, while limiting the need for data exchanges among workers. Once each worker has anonymized its fragment, we combine all the anonymized fragments and obtain an anonymous version of the original dataset. The overall process is overseen by a manager `Man`, which is in charge of: *i)* partitioning the dataset $\mathscr{D}$ in fragments; *ii)* assigning fragments to the different workers; *iii)* collecting and recombining the anonymized fragments from the workers; and *iv)* evaluating the quality of the computed solution. As such, the actual anonymization task (distributed across multiple workers) can be seen as preceded by a *pre-processing step*, where the manager `Man` partitions the original dataset $\mathscr{D}$ in fragments, and assigns fragments to the workers; and followed by a *wrap-up step*, where the anonymized fragments are recombined and the quality of the distributed anonymization is evaluated. Figure 1.2 graphically illustrates our reference scenario, which includes a *(distributed) storage platform*, storing and managing the dataset $\mathscr{D}$ to be fragmented (as well as its anonymized version $\hat{\mathscr{D}}$, after workers have anonymized their fragments), the anonymizing *workers*, and the *manager* `Man` that oversees data fragmentation and allocation

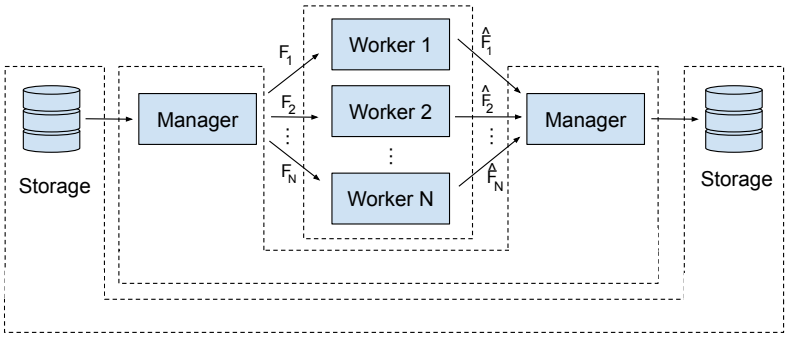


Figure 1.2: Reference scenario

to workers, and recombines the anonymized fragments. In the remainder of this chapter, given a dataset $\mathscr{D}$ over a relation schema including a set QI=$\{a_1,\ldots,a_q\}$ of quasi-identifying attributes, we denote with $\hat{\mathscr{D}}$ its anonymized (i.e., k-anonymous and ℓ-diverse for some input values for k and ℓ) version. Given a tuple $t \in \mathscr{D}$, we denote with $\hat{t}$ the anonymized version of t in $\hat{\mathscr{D}}$, computed by some worker of the system. Similarly, given a fragment F in which $\mathscr{D}$ is partitioned, we denote with $\hat{F}$ the corresponding anonymized version of F (i.e., such that each tuple $t \in F$ has a corresponding anonymized tuple $\hat{t}$ in $\hat{F}$).

The pre-processing phase is crucial for our distributed anonymization, since it produces the fragments of $\mathscr{D}$ that will then be assigned to the workers for anonymization. The first problem to be addressed is then the definition, by the manager, of a fragmentation strategy, regulating which tuples belong to which fragment (and will then be assigned to which worker). An effective strategy, as demonstrated by our experimental results reported in Deliverable D5.4 [PFOR21], is to fragment $\mathscr{D}$ based on the values of (some of the) quasi-identifying attributes $a_1,\ldots,a_h$, in such a way that a tuple t of $\mathscr{D}$ is assigned to a fragment F based on the values of $t[a_1],\ldots,t[a_h]$. To illustrate, consider the dataset in Figure 1.1(a) and suppose to define two fragments F_1 and F_2 based on the values of quasi-identifying attribute `Age`. The manager may define a strategy such that F_1 contains all tuples of $\mathscr{D}$ with values lower than or equal to 39 (i.e., the first and last three tuples of the table), and F_2 the remaining tuples of $\mathscr{D}$. In principle, this would require the manager `Man` to have complete visibility over $\mathscr{D}$ to select the attributes as well as their values for defining fragments. However, in our scenario $\mathscr{D}$ can be too large to fit into the main memory of `Man`. To solve this issue, we propose a strategy in which `Man` can define the conditions that regulate the fragmentation of $\mathscr{D}$ based on a *sample* of the original dataset $\mathscr{D}$, whose size can be dynamically adjusted respecting its storage capabilities. Based on the sample, `Man` can then define a strategy for partitioning $\mathscr{D}$ in fragments and communicate to each worker the conditions that define the tuples in its fragment: with reference to the example above where fragments are defined based on the values of `Age`, `Man` would communicate to workers the value ranges (e.g., with reference to the example above with two workers and two fragments, lower than or equal to 39, and greater than 39) for the tuples of their fragments. Workers can then retrieve from the storage platform the tuples of $\mathscr{D}$ that satisfy such conditions and hence define their fragments, and anonymize them. Figure 1.3 illustrates the pre-processing phase based on a sample of $\mathscr{D}$. Our solution permits to leverage parallel computation of a set of workers for anonymizing large datasets, partitioned among the workers by the manager without requiring complete knowledge of the original datasets themselves, and without affecting the quality of the computed solution as showed by our experimental results reported in Deliverable D5.4 [PFOR21]. We illustrate the

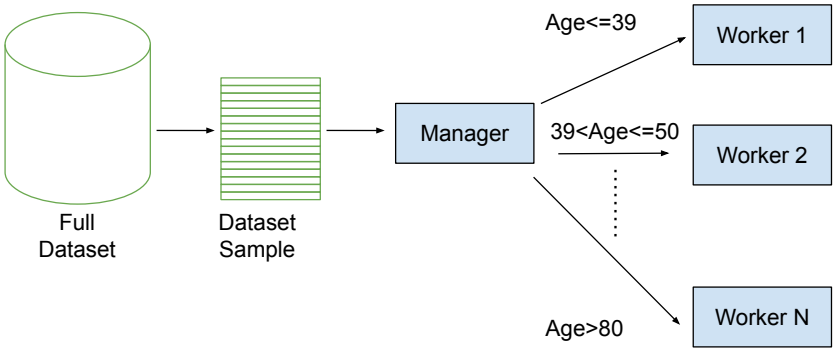


Figure 1.3: Graphical representation of the pre-processing phase

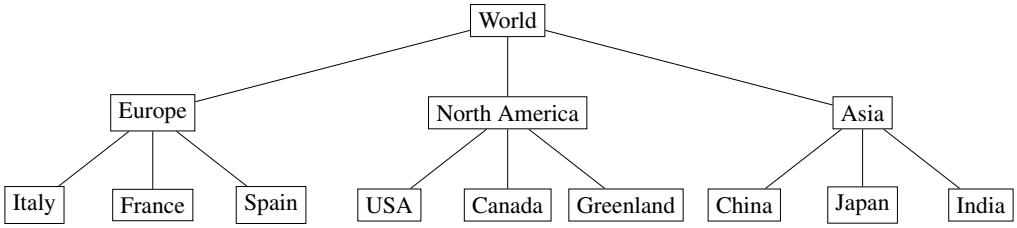


Figure 1.4: Ontology-based generalization hierarchy for attribute `Country`

details of our pre-processing phase in Section 1.4, and of the anonymization and wrap-up phases in Section 1.5.

In our approach, we aim at supporting both numerical and categorical attributes, so to be able to operate with heterogeneous quasi-identifiers. In particular, for categorical attributes we support ontology-based generalization hierarchies for producing semantically-aware generalized (anonymous) data. Figure 1.4 illustrates an example of a generalization hierarchy for attribute `Country`: leaf nodes represent the most specific `Country` values (i.e., those appearing in the original non-anonymized dataset), and their ascendants represent domain-specific generalizations. As an example, considering the dataset in Figure 1.1(a) and the generalization hierarchy in Figure 1.4, values Italy and France may be generalized to value Europe, and USA and Canada to value NorthAmerica.

1.4 Data Pre-processing

In this section, we illustrate the pre-processing phase of our approach. In this phase, the manager `Man` defines how to partition the dataset $\mathscr{D}$ (with quasi-identifying attributes QI) to be anonymized in a set of fragments, which will be independently anonymized by a set $W=\{w_1,\ldots,w_n\}$ of workers. As illustrated in Section 1.3, in our approach this step can operate on a sample D of $\mathscr{D}$, whose size can be tuned depending on the characteristics and storage capabilities of the manager. For the sake of readability, in our discussion we will refer to the problem of fragmenting a generic dataset D with the note that D is a sample of the original dataset $\mathscr{D}$ to be anonymized (note that clearly the quasi-identifying attributes considered for D are those defined for $\mathscr{D}$).

1.4.1 Partitioning Strategies

Selecting a strategy for partitioning the dataset D is crucial in our scenario, since a random partitioning may cause considerable information loss. In fact, a random partitioning may produce fragments that include tuples with heterogeneous values for the quasi-identifier attributes, which would then require a considerable amount of generalization to satisfy the k-anonymity requirement. A more promising strategy is to partition the dataset trying to avoid spreading across fragments tuples that assume similar values for the quasi-identifier attributes. To illustrate, consider quasi-identifying attribute DoB and a dataset D with four tuples $t_1,\ldots,t_4$ that assume for DoB, respectively, values 1980/12/01, 1980/12/02, 1980/05/01, 1980/05/02 and that should be distributed between two workers. Intuitively, if D is partitioned producing a fragment $F_1=\{t_1,t_2\}$, and a fragment $F_2=\{t_3,t_4\}$, it is sufficient for each worker to hide the day of birth of the tuples of the fragment assigned to it to obtain the same (generalized) value, and hence satisfy 2-anonymity. Assigning instead tuples t_1 and t_3 to a worker, and tuples t_2 and t_4 to the other worker would force the generalization to hide (or generalize) also the month of birth to obtain 2-anonymity. In line with this observation, we propose two strategies for partitioning, as follows.

- *Quantile-based.* The dataset D is partitioned according to the values assumed by a quasi-identifying attribute. In particular, given a dataset D with quasi-identifier QI, and a set $W=\{w_1,\ldots,w_n\}$ of workers, quantile-based partitioning selects an attribute a from the QI, and partitions D in n fragments $F_1,\ldots,F_n$ according to the n-quantiles of a in D.

- *Multi-dimensional-based.* The dataset D and its fragments are recursively partitioned in fragments, according to multi-dimensional cuts in a similar way to the Mondrian approach. Given a dataset D with quasi-identifier QI, and a set $W=\{w_1,\ldots,w_n\}$ of workers, multi-dimensional-based partitioning requires different iterations to obtain n fragments. At iteration i, an attribute $a\in$QI (which may be different from the attribute selected at the previous iteration $i-1$) is selected. Each of the fragments obtained at the previous iteration $i-1$ (or, at the first iteration, the dataset D) is then split in two sub-fragments based on the median value assumed by the tuples for a in the fragment being partitioned. The process is executed until (at least, as will be illustrated in the remainder of this section) n fragments are obtained.

Both approaches can be successfully adopted and exhibit different behaviors with respect to the definition of the fragments. The quantile-based partitioning approach ensures balancing among the fragments, since all n fragments will include (approximately) the same number of tuples. On the other hand, its application can be limited by the domain of the attribute a chosen for partitioning (when the number n of workers is larger than the domain of a). On the contrary, while not being limited in applicability by the values of the domain of the attribute chosen for partitioning, the multi-dimensional-based approach may result in workers being assigned twice the work of other workers. This is due to the fact that, at each iteration, this approach doubles the number of fragments of the previous iteration, resulting in the number of created fragments at iteration i to be 2^i. Clearly, the overall number of created fragments is driven by the number of available workers: with n workers, we aim at producing n fragments, so that each fragment can be assigned to a worker, and vice-versa each worker is assigned a fragment. When n is a power of 2, the recursive process can be executed $\log_2 n$ times obtaining exactly n fragments, of (approximately) equal size. When on the contrary n is not a power of 2, there will be an iteration i such that the number 2^i of created fragments is smaller than n (leaving some workers without fragment), and such that the number 2^{i+1} of fragments created at the subsequent iteration would

Q_PARTITION(D, W)

1: **let** a be the attribute over which partitioning D
2: $Ranks := \{rank(t[a]) \mid t \in D\}$ /* ranking of a's values in the ordering */
3: **for each** $i = 1, \ldots, |W|$ **do**
4: **let** q_i be the i^{th} $|W|$-quantile for $Ranks$
5: $F_i := \{t \in D \mid rank(t[a]) \leq q_i \wedge \nexists F_j \neq F_i : t \in F_j\}$

Figure 1.5: Quantile-based partitioning process

M_PARTITION(D, W, i)

1: **let** a be the attribute over which partitioning D
2: $Ranks := \{rank(t[a]) \mid t \in D\}$ /* ranking of a's values in the ordering */
3: **let** m be the median of $Ranks$
4: $F_1 := \{t \in D \mid rank(t[a]) \leq m\}$
5: $F_2 := \{t \in D \mid rank(t[a]) > m\}$
6: **if** $i < \lceil \log_2 |W| \rceil$ **then** /* fragments can be further partitioned */
7: **M_Partition**$(F_1, W, i+1)$
8: **M_Partition**$(F_2, W, i+1)$

Figure 1.6: Multi-dimensional partitioning process

be greater than n (leaving some fragments without an available worker). Aiming at using all n workers, some of the 2^{i+1} fragments obtained at the $i+1$ iteration can then be assigned to the same worker, resulting (to the eyes of the workers) in some fragments being larger than other ones. For instance, assume $W = \{w_1, \ldots, w_7\}$ and $|D| = 1000$. The first iteration ($i = 1$) would partition D in two fragments F_1 and F_2 of approximately 500 tuples each. The second iteration ($i = 2$) would partition F_1 and F_2 in two fragments each (totaling a number of 4 fragments), composed of approximately 250 tuples each. Similarly, the third iteration ($i = 3$) would partition such 4 fragments obtaining a number of 8 fragments, of approximately 125 tuples each. Clearly, since $|W| = 7$, two of these fragments should be assigned to the same worker, resulting in a worker (say, w_1) assigned two fragments (and hence approximately 250 tuples), and the other 6 workers (say, w_2,...,w_7) assigned a fragment only (and hence approximately 125 tuples).

Both the quantile-based and the multi-dimensional-based partitioning approaches illustrated above implicitly require an ordering among the values assumed by the attribute a selected for partitioning, to compute quantiles (for the quantile-based approach) and median values (for the multi-dimensional-based approach). When a is numerical, ordering among its values is naturally defined. When a is categorical, the ordering can be provided by the user or defined according to different strategies. In particular, when a generalization hierarchy $\mathscr{H}(a)$ is defined for attribute a, we consider the ordering in which a's values appear at the leaf level of $\mathscr{H}(a)$. This permits to maintain in each fragment values of a for which the lowest common ancestor (to which they may be generalized, Section 1.5) is lower in the hierarchy, and hence less general, than the ancestor that may be needed by placing in a fragment values of a that are far in the hierarchy. With reference to the hierarchy in Figure 1.4, it is easy to see that placing in a fragment values that are close in the

leaves (e.g., Italy and France) would permit to generalize them with value Europe, while placing in the same fragment values that are further in the leaves (e.g., Italy and China) would require to generalize them to World, that is, the most general value in the hierarchy.

Figure 1.5 illustrates the procedure implementing quantile-based partitioning, executed by the manager `Man` after having retrieved the dataset D. Given the dataset D and a set W of workers, the procedure defines a set *Ranks* including, for each tuple t in D, the rank $rank(t[a])$ of value $t[a]$ in the ordering of the values of a (lines 1–2). It then computes the $|W|$-quantiles for *Ranks*, and defines $|W|$ fragments of D where fragment F_i includes the tuples t of D such that their value $t[a]$ is less than or equal to the i^{th} computed quantile and that do not belong to any other fragment (lines 3–5).

Figure 1.6 illustrates the recursive procedure implementing multi-dimensional-based partitioning, executed by the manager `Man` after having retrieved the dataset D. The procedure takes as input the dataset D, the workers W, and an index i (set to 1 at the first invocation) keeping track of the recursive calls. Given a the attribute over which partitioning D, the procedure defines a set *Ranks* including, for each tuple t in D, the rank $rank(t[a])$ of value $t[a]$ in the ordering of the values of a (line 2). It then computes the median value for *Ranks* (line 3), and defines two fragments F_1 and F_2 such that F_1 includes all the tuples t of D such that the rank of their value $t[a]$ in the ordering is less than or equal to the computed median, and F_2 the remaining tuples (lines 4–5). The procedure is then recursively called on both the computed fragments (lines 7–8) to further fragment them, until the recursive calls index i reaches $\lceil \log_2 |W| \rceil$ (line 6).

In the discussion so far, we have focused on the partitioning of a generic dataset D representing a sample of the original dataset $\mathscr{D}$ to be anonymized. Clearly, for the actual anonymization, workers must retrieve and anonymize the fragments of $\mathscr{D}$. The fragments of $\mathscr{D}$ can simply be defined as the sets of tuples of $\mathscr{D}$ that satisfy the conditions that have created the fragments of D. For instance, consider two fragments F_1 and F_2 computed over a sample D with the multi-dimensional-based approach, with one iteration over attribute `Age` that places all tuples $t \in D$ with `Age` less than or equal to 39 in F_1, and all tuples $t \in D$ with `Age` greater than 39 in F_2. The fragments of $\mathscr{D}$ to be retrieved and anonymized by the workers will be defined in the same way, with all tuples $t \in \mathscr{D}$ with `Age` less than or equal to 39 in F_1 (say, assigned to worker w_1), and all tuples $t \in \mathscr{D}$ with `Age` greater than 39 in F_2 (say, assigned to worker w_2). As illustrated in Section 1.6, the manager can then communicate such conditions to the workers, which can use them to query the storage platform and retrieve the fragments of $\mathscr{D}$ that satisfy them.

1.4.2 Parallelized Multi-dimensional Partitioning

The multi-dimensional-based approach naturally fits a distributed scenario, in which the partitioning is performed in parallel by the workers rather than locally by the manager `Man`. More precisely, in such parallelized multi-dimensional-based approach, the 2^i fragments produced at iteration i (and which should be further split at iteration $i+1$, producing $2 \cdot 2^i$ fragments) are distributed at 2^i workers, each of which will take care of partitioning at iteration $i+1$ the fragment assigned to it. This way, the partitioning operations at a given iteration can be executed in parallel. Figure 1.7 illustrates the steps executed by a worker w upon receiving a fragment to be further partitioned. The parallelized partitioning starts by splitting the entire dataset D, at iteration $i = 1$, at a worker w: given an attribute a chosen for partitioning, the dataset D is split in two fragments F_1 and F_2 where F_1 contains the tuples t of D such that the rank of their value $t[a]$ is less than or equal to the median of the ranks of the values of a in their ordering, and F_2 the remaining tuples

P_PARTITION(D, W, i)
1: **let** a be the attribute over which partitioning D
2: $Ranks := \{rank(t[a]) \mid t \in D\}$ /* ranking of a's values in the ordering */
3: **let** m be the median of $Ranks$
4: $F_1 := \{t \in D \mid rank(t[a]) \leq m\}$
5: $F_2 := \{t \in D \mid rank(t[a]) > m\}$
6: **if** $i < \lceil \log_2 |W| \rceil$ **then** /* fragments can be further partitioned */
7: **let** w_{new} be a new worker in W
8: **assign** $(F_2, i+1)$ **to** w_{new}
9: **P_Partition**$(F_1, W, i+1)$

Figure 1.7: Parallelized multi-dimensional partitioning process

of D (lines 2–5). Rather than locally fragmenting F_1 and F_2 (as done by the manager in the standard multi-dimensional partitioning, Figure 1.6), fragment F_2 is then assigned to a new worker w_{new} (lines 4–5), and the procedure is recursively applied to partition F_1 (by worker w) and F_2 (by worker w_{new}). Similarly to procedure **M_Partition** in Figure 1.6, since at each recursive call the number of fragments in the system doubles (each call splits the original dataset or a fragment in two fragments), given $|W|$ the number of fragments needed, the recursive calls terminate at iteration $i = \lceil \log_2 |W| \rceil$ (line 3).

This parallel partitioning causes an overhead in terms of data transfer, as half of the fragments generated at each step (i.e., in terms of number of tuples, half of the tuples of D) are transferred to additional workers for further partitioning. In particular, given $|D|$ the number of tuples of the dataset D and $|W|$ the number of workers of the system, the overall transmitted data is $1/2 \cdot |D| \cdot (\lceil \log_2 |W| \rceil - 1)$, as can be derived from the recursive calls in Figure 1.7 that cause the transmission of one of the two produced fragments (hence, transmitting at each iteration $1/2 \cdot |D|$ tuples), for $\lceil \log_2 |W| \rceil - 1$ times. On the other hand, it permits to partially parallelize the partitioning process among different workers instead of requiring a local computation at the manager.

1.4.3 Attributes for Partitioning

Regardless of the chosen approach, there is the need of selecting the quasi-identifying attribute a used for partitioning (line 1 in procedures **Q_Partition** in Figure 1.5, **M_Partition** in Figure 1.6, **P_Partition** in Figure 1.7). To this end, several strategies may be adopted [LDR06]. For the quantile-based partitioning approach, we select the attribute $a \in \mathsf{QI}$ that has the largest number of distinct values. The larger the number of distinct values for a quasi-identifying attribute in a fragment, the larger the generalization that should then be applied to anonymize it. By partitioning the dataset D over the attribute with the largest number of distinct values, we obtain fragments where such numerous values have been split by design, helping in reducing the amount of generalization that would then be needed for anonymizing the fragments. For the multi-dimensional-based partitioning approach, following the original Mondrian approach, we propose to select the attribute a that maintains, in the fragment F to be partitioned, the highest representativity (denoted similarity) of the values it assumed in D. If a is numerical, its similarity can be defined based as the ratio between the span of the values in F and the span of the values in the entire dataset D. If a is categorical, its similarity can be defined based as the ratio between the number of distinct values in F

and the number of distinct values in D. More precisely, the similarity $sim(a)$ of a quasi-identifying attribute $a \in \mathsf{QI}$ is defined as follows:

$$sim(a) = \begin{cases} \frac{\max_F\{t[a]\} - \min_F\{t[a]\}}{\max_D\{t[a]\} - \min_D\{t[a]\}} & \text{if } a \text{ is numerical} \\ \frac{\text{count}_F(\text{distinct } t[a])}{\text{count}_D(\text{distinct } t[a])} & \text{if } a \text{ is categorical} \end{cases} \tag{1.1}$$

where $\max_F\{t[a]\}$ and $\min_F\{t[a]\}$ ($\max_D\{t[a]\}$ and $\min_D\{t[a]\}$, respectively) are the maximum and minimum values for a assumed by the tuples in fragment F (in dataset D, respectively); and $\text{count}_F(\text{distinct } t[a])$ ($\text{count}_D(\text{distinct } t[a])$, respectively), is the number of distinct values assumed by attribute a in F (in D, respectively).

Given a quasi-identifying attribute, its similarity according to Equation 1.1 assumes a value in the interval $(0,1]$. Given a fragment F to be fragmented, and a set $\mathsf{QI} = \{a_1, \ldots, a_q\}$ of numerical and categorical quasi-identifying attributes, the attribute $a \in \mathsf{QI}$ chosen for partitioning is the one that has the highest value for its similarity. For example, consider $\mathsf{QI} = \{a_1, a_2, a_3\}$ such that a_1 is categorical and a_2 and a_3 are numerical, and a fragment F to be partitioned. Suppose that: *i)* the distinct values for a_1 are 1000 in D and 100 in F; *ii)* the values for a_2 assume a range of 1000 in D and of 500 in F; and *iii)* the values of a_3 assume a range of 1000 in D and of 700 in F. It follows that $sim(a_1) = 100/1000 = 0.1$; $sim(a_2) = 500/1000 = 0.5$; and $sim(a_3) = 700/1000 = 0.7$. Since $sim(a_3) > sim(a_2) > sim(a_1)$, the attribute chosen for partitioning F in two fragments would be a_3. Clearly, in the first iteration of multi-dimensional-based partitioning the entire dataset D is to be partitioned. For this reason, all quasi-identifying attributes would assume value equal to 1 for similarity (as for all of them the numerator would be equal to the denominator in the similarity computation). In these cases, we select the attribute which exhibits the maximum number of distinct values in the dataset for splitting, like in the case of quantile-based partitioning, large sets of distinct quasi-identifying values among different fragments.

1.5 Data Anonymization and Wrap Up

Once the sample D of the dataset $\mathscr{D}$ has been partitioned in fragments, the actual anonymization process can start. The manager `Man` communicates to each worker w the conditions that define its fragment, and w retrieves from the storage platform the tuples of $\mathscr{D}$ that satisfy them, hence obtaining the fragment of $\mathscr{D}$ it has to anonymize. Each worker then anonymizes its fragment and computes the information loss on it. The wrap-up phase collects all the anonymized fragments and stores them to the storage platform, and computes the overall information loss.

1.5.1 Data Anonymization

In this step, each worker anonymizes the fragment assigned to it, executing (without the need of interacting with other workers) an extended version of the Mondrian original approach. In particular, the Mondrian partitioning is performed considering the requirements of both k-anonymity and ℓ-diversity (in contrast to the original approach, which only considered k-anonymity [LDR06]).

Figure 1.8 illustrates the anonymization process executed by each worker for anonymizing the fragment assigned to it. The recursive partitioning (lines 3–10), which operates according to the same logic of the multi-dimensional partitioning in the pre-preprocessing phase (Section 1.4), terminates when no further partitioning could be computed without violating k-anonymity or ℓ-diversity (lines 1–2), in which case the fragment is anonymized by generalizing the values for the

ANONYMIZE(F)
1: **if** no partitioning can be done without violating k-anonymity or ℓ-diversity **then**
2: **generalize** F[QI]
3: **else**
4: **let** a be the attribute for partitioning
5: $Ranks := \{rank(t[a]) \mid t \in F\}$ /* ranking of a's values in the ordering */
6: **let** m be the median of $Ranks$
7: $F_1 := \{t \in F \mid rank(t[a]) \leq m\}$
8: $F_2 := \{t \in F \mid rank(t[a]) > m\}$
9: **Anonymize**(F_1)
10: **Anonymize**(F_2)

Figure 1.8: Anonymization process for a fragment F

quasi-identifying attributes. The attribute a over which partitioning (line 4) is chosen adopting the same metrics illustrated for the definition of the fragments of the sample D (Section 1.4.3), that is, choosing the quasi-identifying attribute with maximum value for its similarity (Equation 1.1). Clearly, since during the anonymization phase each worker w has visibility only on the fragment F assigned to it, the similarity values computed by it are defined with respect to the span and number of distinct values (i.e., the denominators of Equation 1.1) in F (in contrast to the entire dataset D in Equation 1.1). Since also in the first partitioning of the anonymization of a fragment all quasi-identifying attributes would have their similarity equal to 1, for the first partitioning our approach selects the attribute that has the highest number of distinct values in the considered fragment.

As for generalization, our extended version of Mondrian supports different strategies, suited for different kinds of data, as follows.

- *Ontology-based generalization hierarchies*: applicable to categorical data, the generalization of an attribute a follows the generalization hierarchy $\mathscr{H}(a)$ defined for a (Section 1.3). Given a set of tuples in a fragment F that should be anonymized, and a hierarchical attribute a in the QI that needs to be generalized with a custom taxonomy $\mathscr{H}(a)$, the original tuples in F are replaced with generalized ones that assume, for a, the closest ancestor in $\mathscr{H}(a)$ of all the values of a appearing in F. To illustrate, consider the dataset in Figure 1.1(a). Its anonymized version in Figure 1.1(c) is obtained generalizing attribute `Country` according to the generalization hierarchy in Figure 1.4. For example, values *Italy*, *Italy*, and *France* of the first three tuples are generalized to their closest ancestor *Europe* in the hierarchy. Values *Italy*, *France*, and *Canada* of the fourth, fifth, and sixth tuples are generalized to value *World* (their closest ancestor, which is also the root of the hierarchy). Values *USA*, *USA*, and *USA* of the last three tuples can remain as such, since they already are the same value.

- *Common prefix:* applicable to categorical data, the generalization of an attribute a considers the values to be generalized as strings and is performed by replacing the original values with generalized ones that only include their common prefix (thus redacting the characters that differ). For instance, values 10010, 10020, 10030 for attribute `ZIP` can be generalized to the value $100**$, maintaining their common prefix 100 and redacting (in this example, replacing with value $*$) the last two digits.

- *Set definition:* applicable to both categorical and numerical data, the generalization of an attribute a is performed by grouping its values in sets. For instance, values *Italy*, *France*, and *Spain* for (categorical) attribute `Country` can be generalized to the set $\{Italy, France, Spain\}$. Similarly, values 50, 60, 85 for (numerical) attribute `Age` can be generalized to the set $\{50, 60, 85\}$.

- *Interval definition:* applicable to numerical data, the generalization of an attribute a is performed by grouping its values in an interval that contains them. While the smallest interval containing all the values to be generalized is the most natural choice, we note that larger (and possibly pre-defined) intervals may also be adopted. Note that this generalization is different from the one following the set definition: while the set definition explicitly maintains all (original) values that are generalized in the set, here only the endpoints of the interval are maintained, as the generalization returns an interval without specifying which intermediate values in the interval have been generalized. To illustrate, consider the dataset in Figure 1.1(a). Its anonymized version in Figure 1.1(c) is obtained generalizing attribute `Age` by grouping its values in intervals. For example, values 25, 25, and 30 of the first three tuples are generalized to interval $[25, 30]$.

1.5.2 Wrap Up and Information Loss Assessment

In the wrap-up phase, the anonymized fragments are collected by the manager and stored to the storage platform, and the information loss caused by anonymization is assessed. In particular, information loss is evaluated by each worker on its anonymized fragment $\hat{F}$, and the assessments are then collected and combined by the manager for evaluating the information loss on the entire anonymized dataset $\hat{\mathscr{D}}$.

We evaluate information loss adopting different metrics proposed in the context of data anonymization, as we illustrate in this section. For the sake of readability, we illustrate the metrics as applied to the anonymized version $\hat{D}$ of a generic dataset D, with the note that in our proposal each worker applies them to the anonymized version of the fragment of $\mathscr{D}$ assigned to it.

- *Discernibility Penalty* (DP). The first metric we adopt is a standard metric, also adopted by the original Mondrian approach [LDR06] and in the literature [BA05], and assigns a *penalty* to each original tuple of D that reflects the number of tuples in $\hat{D}$ that are indistinguishable from it (that is, the size of the equivalence class E to which the tuple belongs). More precisely, given E the equivalence classes in $\hat{D}$, the Discernibility Penalty metric $\mathsf{DP}(\hat{D})$ for $\hat{D}$ is computed as:

$$\mathsf{DP}(\hat{D}) = \sum_{E \in \hat{D}} |E|^2 \tag{1.2}$$

- *Normalized Certainty Penalty* (NCP). The second metric we adopt aims at quantifying penalties depending on the amount of generalization applied to the values of the quasi-identifying attributes, and assigns larger penalties for the application of more generalization [XWP+06]. It is applicable to numerical attributes generalized in intervals, and to categorical attributes for which a generalization hierarchy exists. More precisely, given a quasi-identifying numerical attribute a and a tuple t in D generalized in $\hat{D}$ so that $t[a]$ is generalized to an interval $[v_{min}, v_{max}]$, the normalized certainty penalty $\mathsf{NCP}_a(\hat{t})$ of $\hat{t}$ for a is

computed as:

$$\mathsf{NCP}_a(\hat{t}) = \frac{v_{max} - v_{min}}{Range(a)} \tag{1.3}$$

where $Range(a)$ is the range of the values assumed by a.
For categorical data, given a quasi-identifying categorical attribute a and a tuple t in D generalized in $\hat{D}$ in a tuple $\hat{t}$ such that $\hat{t}[a]$=$\hat{v}$, the normalized certainty penalty $\mathsf{NCP}_a(\hat{t})$ of $\hat{t}$ for a is computed as:

$$\mathsf{NCP}_a(\hat{t}) = \frac{|Ind(\hat{v})|}{|Dom(a)|} \tag{1.4}$$

with $Ind(\hat{v})$ the set of other values in $Dom(a)$ that could be generalized to $\hat{v}$ (i.e., the number of other leaves of the generalization hierarchy for a that are also descendants of $\hat{v}$).
Given a generalized dataset $\hat{D}$ with quasi-identifier QI composed of both numerical and categorical attributes, the normalized certainty penalty of $\hat{D}$ is then computed as:

$$\mathsf{NCP}(\hat{D}) = \sum_{\hat{t} \in \hat{D}} \sum_{a \in \mathsf{QI}} \mathsf{NCP}_a(\hat{t}) \tag{1.5}$$

with $\mathsf{NCP}_a(\hat{t})$ the normalized certainty penalty of $\hat{t}$ for a computed according to Equations 1.3 and 1.4 depending on whether a is numerical or categorical.

The information loss values computed by each worker on its fragment are collected by the manager, which can then combine them (through a simple sum) to assess the information loss of the entire anonymized dataset $\hat{\mathscr{D}}$. Our metrics permit to assess information loss with different approaches. The DP metric assigns penalties depending on the size of equivalence classes: the larger an equivalence class, the larger the penalty regardless of how the quasi-identifying attributes within the equivalence class have been generalized. Consider QI=`{ZIP}` and two equivalence classes E_1 and E_2 such that $|E_1|$=$|E_2|$, and such that `ZIP` has been generalized by removing the last digit only in E_1, and all digits in E_2. The DP metric would assign to E_1 and E_2 the same penalty, equal to $|E|^2$, even though E_1 intuitively carries more information than E_2 where `ZIP` is actually suppressed. On the other hand, the NCP metric assigns penalties depending on the actual generalized values: informally, the 'more generalized' a value, the larger the penalty. For example, consider QI=`{Age}`: with NCP, the generalization of value 50 in the range $[40,60]$ would be assigned a smaller penalty than its generalization in the range $[20,80]$.

1.6 Implementation

In this section, we provide an update on architectural design and deployment of the tool illustrated in Deliverable D5.4 [PFOR21] for enforcing distributed anonymization. The implementation is available at `https://github.com/mosaicrown/mondrian`.

Our implementation of a system for distributed anonymization is based on an *Apache Spark* cluster, whose nodes perform the different phases (pre-processing, anonymization, and wrap-up) illustrated in the previous sections. The dataset $\mathscr{D}$ to be anonymized can be stored on any storage platform (centralized or distributed) reachable by Apache Spark with a URL. The overall architecture of our system is illustrated in Figure 1.9. The Spark cluster is composed of a *Spark Cluster Manager*, which coordinates the cluster, and of a set W of *Spark Workers*, which perform the tasks assigned to them by the Cluster Manager. The distributed anonymization process illustrated in the

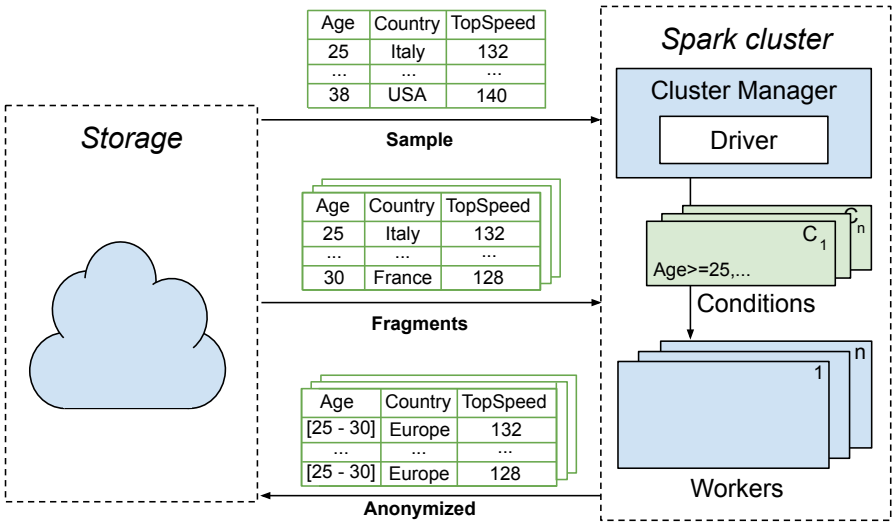


Figure 1.9: Architecture and working of our distributed anonymization system

previous sections occurs within the Spark cluster: in particular, the cluster is in charge of retrieving a sample D of the dataset $\mathscr{D}$, fragmenting $\mathscr{D}$ based on the fragmenting conditions evaluated on D (Section 1.4), assigning the fragments to the workers, executing the anonymization at the workers, and wrapping up by retrieving the anonymized fragments and the computed information loss from the workers. We developed our distributed Spark anonymization application in Python to leverage the Pandas framework, which can be conveniently used for managing large datasets.

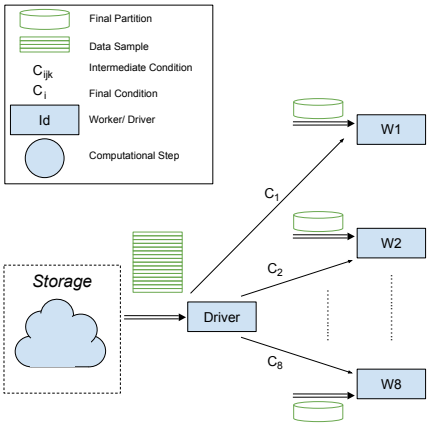
The manager `Man` in charge of coordinating the overall process is represented in our architecture by the *Spark Driver*, responsible for calling the *Spark Context* in the the user-written code (implementing the quantile- and multi-dimensional-based partitioning, and acting as a coordinator in case of parallelized multi-dimensional-based partitioning), and for translating this code into jobs that are actually executed on the cluster. Jobs are further divided into smaller execution units, called *tasks*. Once pre-processing is completed, the actual anonymization is divided in tasks that are then executed by the various workers. Note that the Spark Driver and the Spark Cluster Manager are not necessarily hosted on the same node of the cluster.

The overall working of our system starts with the Spark Driver retrieving from the storage platform a sample D of $\mathscr{D}$ that fits into its memory. The pre-processing phase is then executed differently depending on whether it follows the quantile-based, multi-dimensional-based, or parallelized multi-dimensional-based approach, as follows.

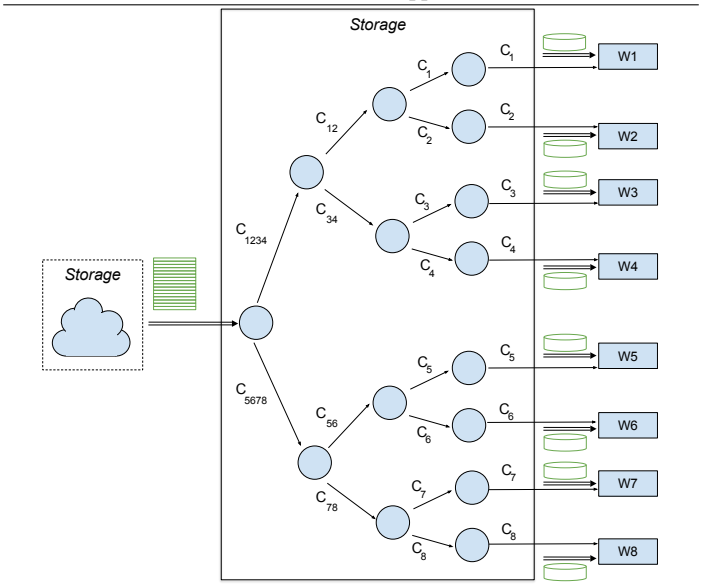
- **Quantile-based and multi-dimensional-based partitioning** (Section 1.4.1). The fragmentation of D is locally executed at the Spark Driver, which executes the partitioning processes in Figures 1.5 and 1.6. The Driver keeps track of the conditions that define the fragments computed over the retrieved sample D, and defines a set of $|W|$ Spark tasks corresponding to the anonymization of a fragment of $\mathscr{D}$, to be sent to the workers. Each task includes the conditions that define the fragment to be anonymized, which will be used by the assigned worker to retrieve its fragment of $\mathscr{D}$ from the storage platform. Once the Spark Driver has terminated the pre-processing operations and defined the fragment anonymization tasks, the Spark Cluster Manager selects the workers that will be involved in the actual anonymization. The Spark Driver produces a scheduling of the tasks and sends them to the Spark Workers identified by the Spark Cluster Manager. Figures 1.10(a)–(b) illustrate the involve-

ment of the Spark components in the fragmentation process of a sample D of a dataset $\mathscr{D}$ assuming to produce 8 fragments to be then assigned to 8 workers adopting quantile- and multi-dimensional-based partitioning. Double arrows correspond to the transfer of data, while single arrows represent the communication of a fragmenting condition c (in the multi-dimensional partitioning approaches, final conditions that define the fragments of $\mathscr{D}$ have a single index c_i, while intermediate conditions generated by the recursive iterations have multiple indices $c_{i...k}$). Green rectangles correspond to the sample D of the dataset $\mathscr{D}$, while the database icon corresponds to a fragment F of $\mathscr{D}$. As illustrated in Figure 1.10(a), in the quantile-based partitioning the Spark Driver retrieves the sample D of $\mathscr{D}$ from the storage platform and locally partitions D in fragments (8 fragments in the example in the figure). The 8 workers then receive the conditions defining the 8 fragments. Multi-dimensional-based partitioning (Figure 1.10(b)) operates in a similar way, with the difference that the Driver executes the recursive process in Figure 1.6, defining at each iteration a fragmenting condition, and the final conditions defining the fragments of $\mathscr{D}$ are the conjunction of those recursively computed in the iterations of the partitioning process. For example, suppose that the first partitioning of a sample D places all tuples of D with values for `Age` less than or equal to 39 in F_1, and the remaining ones in F_2. With reference to Figure 1.10(b), c_{1234} is "`Age` $\leq$ 39", and c_{5678} is "`Age` $>$ 39". Suppose that the subsequent recursive partitioning is done on attribute `Country`, and that it fragments both F_1 and F_2 by placing all tuples with `Country` equal to *Italy* and *France* in a fragment, and equal to *USA* and *Canada* in the other fragment. Conditions c_{12}, c_{34}, c_{56}, and c_{78} would then be defined as: c_{12} = "(`Age` $\leq$ 39) AND (`Country` IN {*Italy*, *France*})"; c_{34} = "(`Age` $\leq$ 39) AND (`Country` IN {*USA*, *Canada*})"; c_{56} = "(`Age` $>$ 39) AND (`Country` IN {*Italy*, *France*})"; and c_{78} = "(`Age` $>$ 39) AND (`Country` IN {*USA*, *Canada*})".

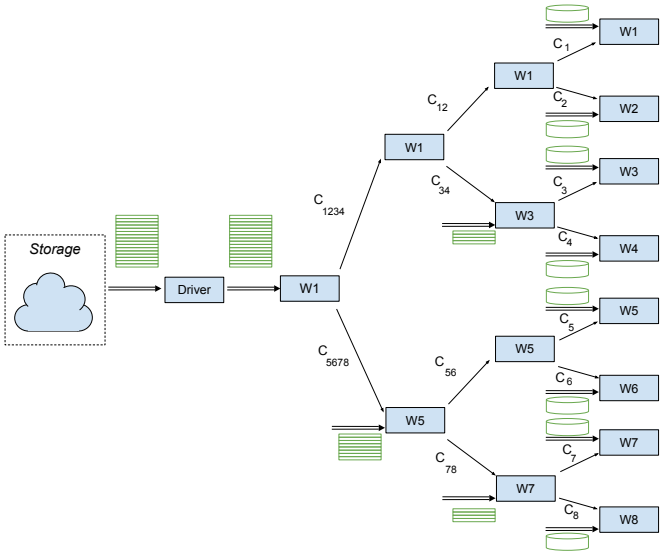
- **Parallelized multi-dimensional partitioning** (Section 1.4.2). The fragmentation of D is executed by a set of Spark Workers (again selected, like in the previous case, by the Cluster Manager). The Driver collects the sample D of $\mathscr{D}$ and coordinates the involvement of the different workers in the fragmentation process. In particular, the Driver sends D to the first Spark Worker w_1, which fragments D in two fragments F_1 and F_2 and returns to the Driver the conditions that define them. The Driver then involves a second worker w_2, and communicates to w_1 and w_2 which one, between F_1 and F_2, they are assigned to for the next partitioning iteration (by communicating to each of them the conditions that define its fragment). The newly added worker w_2 retrieves from the Driver the fragment assigned to it, and w_1 (which produced and therefore stores both F_1 and F_2) discards the one assigned to w_2. Both workers then proceed with partitioning the fragment assigned to them, obtaining a total of 4 fragments, and communicate the conditions that define them to the Driver, which involves other workers. This process is repeated until the desired number of fragments is obtained. While this procedure demands a larger volume of transferred data, compared to the other approaches, it also permits to avoid a cold-start of the procedure, allowing the parallelization of the pre-processing phase as soon as possible. Figure 1.10(c) illustrates the involvement of the Spark components for computing the 8 fragments of a sample D in case of parallelized multi-dimensional partitioning. Differently from Figure 1.10(b), here the fragmentation is spread across the different workers. At each iteration of the fragmentation process, only the newly added worker retrieve the tuples composing their sample fragments for further partioning (as represented by the double arrows). For the sake of readability, in



(a) Quantile-based approach



(b) Multidimensional-based approach



(c) Parallelized multidimensional-based approach

Figure 1.10: Fragmentation in Spark according to the possible strategies

the figure we have represented the transfer of the conditions as directly happening between workers, while the communication is always mediated by the Spark Driver.

Regardless of the chosen partitioning strategy, once a Spark worker receives its anonymization task, it retrieves from the storage platform the fragment of $\mathscr{D}$ satisfying the fragmenting conditions included in the task, and anonymizes it by executing our version of Mondrian (Figure 1.8, Section 1.5). Spark Workers also compute the information loss on their fragments (Equations 1.2 and 1.5, Section 1.5), completing their tasks. At the end of this process, Spark Workers send to the storage platform the anonymized fragments through the Spark Driver, which combines the information loss evaluations from the workers to evaluate the quality of the overall solution.

Since we aim to create a solution that can be easily deployed in a cloud infrastructure such as AWS or Google Cloud, for the deployment of our application we have opted for a multi-container application, leveraging in particular Docker containers. We therefore deployed the Spark architecture in Figure 1.9 with a set of Docker containers composed of: *i)* one container for the Spark Driver (which, as previously discussed, is responsible for the definition and assignment of the anonymization tasks to the workers, and the collection of the results); *ii)* one container for the Spark Cluster Manager (responsible for selecting the workers); *iii)* a variable number of containers for the Spark Workers (which perform the anonymization tasks and compute information loss); and *iv)* an additional container used to expose a *Spark History Server* (for additional information about the task scheduling and assignment performed by Spark).

Docker containers are spawned in our implementation with Docker Compose and, to manage the distribution of the containers to the nodes (machines) of the Spark clusters, different orchestrator tools (e.g., Kubernetes) may be adopted. In our implementation, we leverage Docker Swarm due to its simplicity. With Docker Swarm, nodes of the cluster can operate as *managers* or *workers*, so that the swarm manager is in charge of distributing the workload on the various swarm workers, which in turn are used to spawn the Docker containers modeling the components of Spark. Note that swarm worker nodes can be used to spawn multiple containers. Figure 1.11 graphically represents an example of the Docker Swarm distribution of our containers to the nodes of the Spark cluster. Each white rectangle corresponds to a node in the cluster, and the blue boxes within nodes represent Docker containers. In particular, one node acts as swarm manager, while all other nodes act as swarm workers coordinated by it. One of the swarm workers used for anonymization is designed to spawn one container for the Spark Cluster Manager and one for the Spark Driver, while the other swarm workers spawn the needed set of containers for the Spark workers.

1.7 Experimental Results

We performed a set of experiments to evaluate the scalability and applicability of our approach for distributed anonymization. Our experiments, executed enforcing our extended version of Mondrian with varying values of k and ℓ and with the different partitioning approaches, aim at evaluating the time needed for anonymizing a dataset with our extended version of Mondrian compared to a baseline where the same dataset is anonymized with the traditional centralized Mondrian approach. A preliminary version of the experimental evaluation is reported in Deliverable D5.4 [PFOR21].

1.7.1 Experimental Settings

Our experiments have been with the following settings.

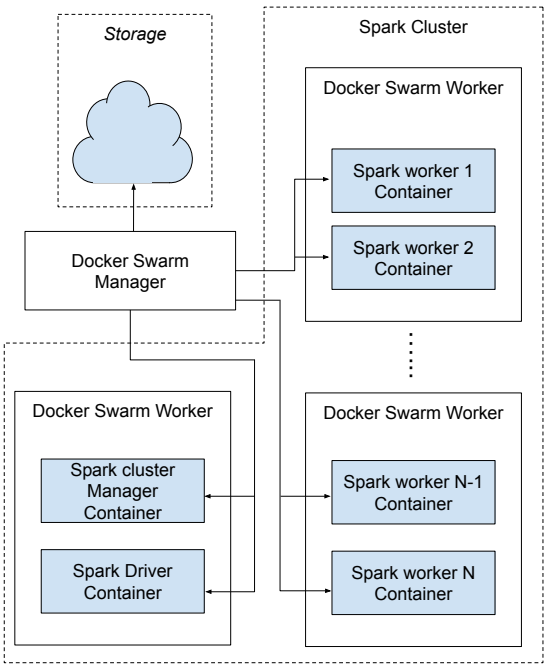


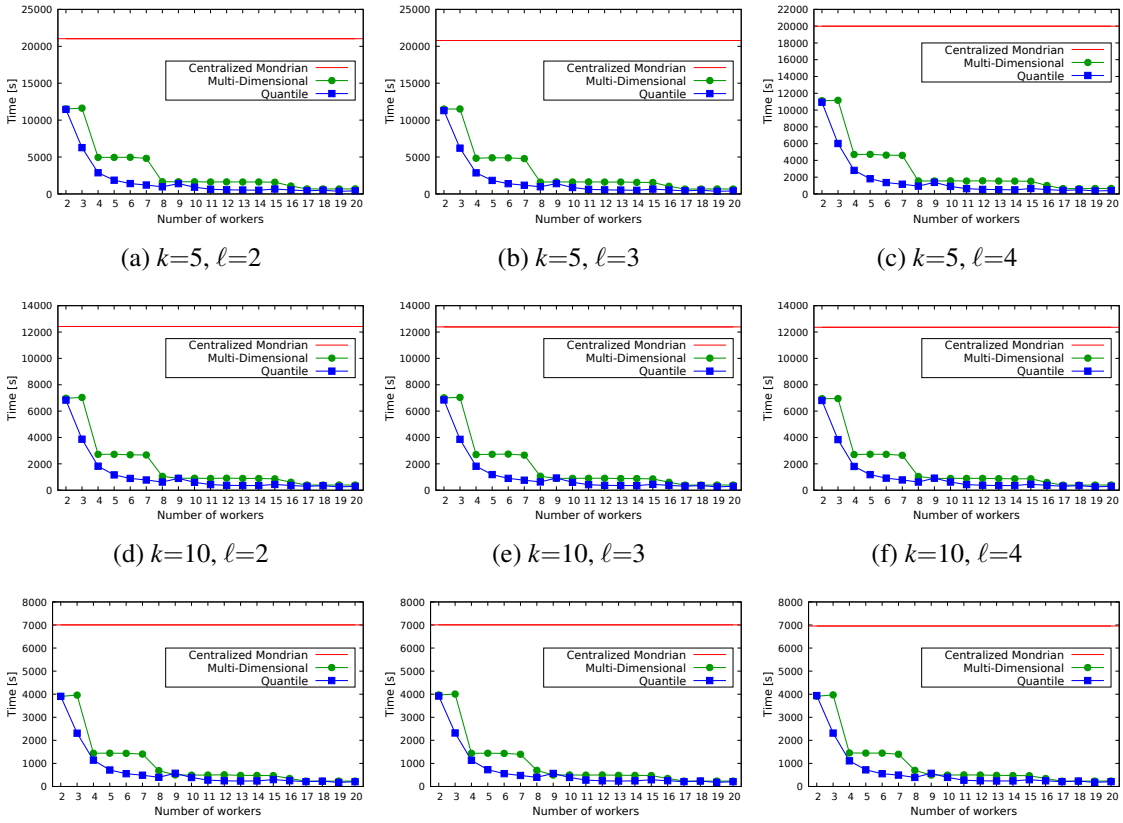
Figure 1.11: Container distribution in a cloud environment

Server specifications and cloud deployment. Our solution does not require a specific underlying cloud environment. To obtain a reproducible environment, for our experiments we have simulated a cloud environment leveraging Docker Compose. We run our experiments on a machine equipped with an AMD Ryzen 3900X CPU (12 physical cores, 24 logical cores), 64 GB RAM and 2 TB SSD, running Ubuntu 20.04 LTS, Apache Spark 3.0.1, Hadoop 3.2.1, and Pandas 1.1.3. Each worker is equipped with 2GB of RAM and 1 CPU core. The baseline executing the traditional centralized Mondrian approach is run on a worker equipped with 1 core and without RAM limitations. To further prove the applicability and scalability of our solution in a real distributed environment, we have also deployed it on the Openstack Cloud Computing environment offered by Cineca [1] as well as on Amazon Elastic Compute Cloud (*t2.medium* instances equipped with 2 cores, 4GB of RAM, and 8GB of gp2 SSD, running Ubuntu 20.04 LTS, and located in the *us-east-1* region).

Dataset. Our experiments have been executed on the *ACS PUMS USA 2019* [U.S19] dataset, composed of 3,239,554 rows. Each tuple of the dataset represents an individual respondent through attributes `ST`, `OCCP`, `AGEP`, and `WAGP`, representing respectively the respondent's US State of residence, occupational status (expressed with a numeric code), age, and annual income. We considered `ST`, `OCCP`, `AGEP` as the quasi-identifier for this dataset, and `WAGP` as the sensitive attribute. While `OCCP`, `AGEP`, and `WAGP` are numeric attributes, `ST` is a categorical attribute with a generalization hierarchy $\mathscr{H}(\texttt{ST})$ defined according to the criteria adopted by the US Census Bureau[2], such that States are at the leaf level of the hierarchy and are grouped in Divisions, which are in turn grouped in Regions. For example, States NJ, NY, and PA (leaf level) can be generalized to MiddleAtlantic (which is their parent in the hierarchy), which in turn can be generalized to Northeast, which in turn can be generalized to US (which is the root of the hierarchy).

[1] `https://cloud.hpc.cineca.it/`

[2] `https://www2.census.gov/geo/pdfs/maps-data/maps/reference/us_regdiv.pdf`



(a) k=5, ℓ=2 (b) k=5, ℓ=3 (c) k=5, ℓ=4

(d) k=10, ℓ=2 (e) k=10, ℓ=3 (f) k=10, ℓ=4

(g) k=20, ℓ=2 (h) k=20, ℓ=3 (i) k=20, ℓ=4

Figure 1.12: Runtime results, ACS PUMS USA 2019

Storage platform. For our experiments, we leveraged *Hadoop Distributed File System (HDFS)* as the distributed storage platform for storing the datasets to be anonymized. We realized the HDFS cluster leveraging Docker containers, with one container for the Hadoop Namenode (responsible for the cluster management); and multiple containers for the Hadoop Datanodes (responsible for storing data and servicing read and write requests).

1.7.2 Results

We discuss the results of a set of experiments evaluating the computational time of our approach varying the number of workers, to the aim of assessing the scalability of our approach.

To prove that the scalability of our approach is not affected by specific configurations, we tested different configurations for k and ℓ. In particular, we executed our distributed anonymization approach with values of k equal to 5, 10, and 20, and with values of ℓ equal to 2, 3, 4. A centralized baseline is computed for each configuration. Figure 1.12 reports the runtime needed by our approach (considering both quantile-based partitioning and multi-dimensional-based partitioning) for anonymizing the ACS PUMS USA dataset compared to the centralized baseline.

From the curves in the figure it is possible to see that, as expected, the execution time needed for anonymization decreases when the number of workers grows, with a saving with respect to the centralized baseline that ranges from 43% to 98%, thus confirming the scalability of our distributed approach. It is interesting to note that the quantile-based partitioning and the multi-dimensional-based partitioning entail the same execution time when two workers are used (in-

deed, both approaches would perform the same partitioning) and that, while the quantile-based partitioning sees an improvement of the runtime every time an additional worker is computed, the multi-dimensional-based partitioning improves its performance when the number of workers is a power of 2. As expected, this is due to the fact that when the number of workers is not a power of 2, some workers are assigned larger fragments (as discussed in Section 1.4.1). On the other hand, as shown by the next set of experiments, multi-dimensional-based partitioning ensures a reduced information loss if compared to quantile-based partitioning. In the preliminary results illustrated in Deliverable D5.4 [PFOR21], also appeared in [DFF+21b], we noted that the centralized version was more efficient than our proposed distributed version when the number of workers was low (2 or 3 in our experiments). This was due to the constant initialization time paid by the distributed implementation for setting distribution and interoperation among workers, and by the different libraries used by the centralized implementation (NumPy) and by the distributed implementation (Spark APIs) in the smaller dataset used (composed of 500,000 tuples); as demonstrated in the experiments reported in this chapter, this effect does not hold for large datasets.

2. Secure Multi-Party Computation of Differentially Private Statistics

In this chapter, we describe the progress of the collaborative computation techniques developed in MOSAICrOWN. In particular, this chapter illustrates an extension to the two-party secure computation for rank-based statistics detailed in Deliverable D5.4 [PFOR21]. We extend the applicability to *multiple* parties that collaboratively and securely compute a broader class of differentially private statistics. In more detail, the extension also supports rank-based statistics but also further encompasses any statistic that can be expressed as a decomposable aggregate function as used in MapReduce-style frameworks. In the following, we consider the problem of distributed private learning; specifically, how multiple users can compute rank-based statistics over their sensitive data, with high accuracy, a strong privacy guarantee, and without resorting to trusted third parties. We use *differential privacy* (DP) [Dwo06, DMNS06], a rigorous privacy notion, restricting what can be inferred about any individual in the data, used by Google [EPK14, BEM+17], Apple [App16, Tea17], Microsoft [DKY17] and the US Census bureau [Abo18].

In Section 2.2, we describe preliminaries for our protocol. In Section 2.3 we explain our protocol and introduce definitions. For illustration purposes we will focus our description on rank-based description, especially the median, as in Deliverable D5.4 [PFOR21] and detail additional applications in Section 2.3.1. We present our protocol and implementation details for the secure multi-party computation of the differentially private median in Section 2.4. We provide a detailed performance evaluation in Section 2.5, describe related work in Section 2.6 and conclude in Section 2.7.

2.1 State of the Art and MOSAICrOWN Innovation

In this section, we briefly summarize the state of the art and discuss the innovation produced during MOSAICrOWN in relation to existing work.

2.1.1 State of the Art

Previous work on DP median computation either require a large number of users to be accurate [DL09, STU17, GSX18], rely on a trusted third party [NRS07, McS09], or cannot scale to large universe or data set sizes [PL15, EKM+14]. We present a novel alternative that is superior in accuracy, requires no trusted party, and is efficiently computable. Our protocol provides high accuracy even for a small number of users. Note that small sample size is the most challenging regime for DP [NRVW20], as the noise can easily overwhelm the (statistical) signal for a small number of data samples. Even Google's large-scale data collection (billions of daily reports via [EPK14]) is insufficient if the statistical value of interest is not a heavy hitter [BEM+17], e.g., the

median. Related work is further discussed in Section 2.6 after we presented our solution in detail in Section 2.4.

2.1.2 MOSAICrOWN Innovation

We present a *secure multi-party computation* (MPC) of the *exponential mechanism* [MT07] for *decomposable aggregate functions*. Such functions, as used in MapReduce-style algorithms [DG08], allow efficient aggregation in parallel over distributed datasets, and application examples include convex loss functions and rank-based statistics. The exponential mechanism can implement any differentially private algorithm by computing selection probabilities for all possible output elements. Its computation complexity is linear in the size of the data universe, i.e., the domain of possible values for the data, and efficiently sampling it is non-trivial [MT07, DR14]. Also, the exponential mechanism requires exponentiations, increasing the MPC complexity. However, as it is a universal mechanism, a scalable, secure implementation can be widely applied. Eigner et al. [EKM+14] also implement the exponential mechanism in MPC. They compute the exponential function with MPC, whereas we provide a more efficient alternative for decomposable functions. Their approach, while more general, is only practical for a universe size of 5 elements, whereas our protocol is sublinear in the size of the universe, and is capable of processing billions of elements. We achieve this via divide-and-conquer and optimizing our protocol for decomposable functions that enable efficient alternatives to expensive secure computation of exponentiations [DFK+06, AS19, Kam15, ABZS13].

In summary, the innovation developed within MOSAICrOWN are protocols for securely computing the differentially private statistics.

- The first innovation is that the presented protocols allow high accuracy even in the most challenging regime with small datasets and potentially large universe sizes.
- The second innovation is that the custom-made cryptographic protocols leverage insights from the underlying statistical problems to be very efficient compared to general secure multi-party solutions, i.e., even in a wide area network, the solutions achieve running time of seconds (with relaxed group privacy for large datasets as in Deliverable D5.4 [PFOR21]) and practical running time of few minutes (without relaxation as in this chapter).
- The third innovation lies in the scalability of the approach which has a computation complexity that is sublinear in the data universe size (and not linear as typically found for probabilistic selection via the exponential mechanism).

Some of the results obtained by MOSAICrOWN have been published in [BK20a, BK20b, BK21].

2.2 Preliminaries

In the following, we introduce preliminaries for differential privacy and secure multi-party computation.

We consider a set of input parties $\mathscr{P} = \{P_1, \ldots, P_n\}$, where party P_i holds a datum d_i, and D denotes their combined dataset. We model a dataset as $D = \{d_1, \ldots, d_n\} \in U^n$ with underlying *data universe* U. We also consider m semi-honest *computation parties* who run the computation on behalf of the input parties. In more detail, our evaluation considers $3, 6$, and 10 as the number of computation parties. To simplify presentation, we assume the size n of D to be even, which can be ensured by padding. Then, the median's position in sorted D is $n/2$.

2.2.1 Differential Privacy

Differential privacy (DP), introduced by Dwork et al. [Dwo06, DMNS06], is a strong privacy guarantee restricting what a mechanism operating on a sensitive dataset can output. Informally, when the input dataset changes in a single element, the effect on the output is bounded. The formal definition is as follows:

Definition 1 (Differential Privacy)**.** *A mechanism $\mathscr{M}$ satisfies ε-differential privacy, where $\varepsilon \geq 0$, if for all neighboring datasets $D \simeq D'$, i.e., datasets differing in a single entry, and all sets $S \subseteq Range(\mathscr{M})$*

$$Pr[\mathscr{M}(D) \in S] \leq \exp(\varepsilon) \cdot Pr\big[\mathscr{M}(D') \in S\big],$$

where $Range(\mathscr{M})$ denotes the set of all possible outputs of mechanism $\mathscr{M}$.

The above definition holds against an unbounded adversary, however, due to our use of cryptography we assume a computationally bounded adversary.

The original definition of Differential Privacy considers the central model with unbounded adversaries [Dwo06, DMNS06] (see Definition 1), later work expanded it to a distributed setting [DKM$^+$06, KLN$^+$11], and considered computationally-bounded parties [MPRV09].

We consider multiple computationally-bounded, semi-honest parties performing a joint secure computation realized with (t,m)-secret sharing. The following definition from [EKM$^+$14] fits our setting, where $\texttt{VIEW}^p_\Pi(D)$ denotes the view of party p during the execution of protocol Π on input D, including all exchanged messages and internal state, and λ is a security parameter:

Definition 2 (Distributed Differential Privacy)**.** *A randomized protocol Π implemented among m computation parties $\mathscr{P} = \{P_1, \ldots, P_m\}$, achieves* distributed differential privacy *w.r.t. a coalition $C \subset \mathscr{P}$ of semi-honest computation parties of size t, if the following condition holds: for any neighbors D, D' and any possible set S of views for protocol Π,*

$$Pr\big[\mathtt{VIEW}^C_\Pi(D) \in S\big] \leq \exp(\varepsilon) \cdot Pr\big[\mathtt{VIEW}^C_\Pi(D') \in S\big] + negl(\lambda).$$

The definition can be expanded to a malicious setting by replacing the semi-honest parties $\mathscr{P}$ and semi-honestly secure protocol Π with malicious parties and a maliciously secure protocol. Note that the negligible function $\text{negl}(\lambda)$ can be omitted for protocols providing information-theoretic security (such as standard secret sharing), however, our implementation in SCALE-MAMBA provides computational security (due to the online phase as described in Section 2.2.2).

Randomization is essential for differential privacy to hide an individual's inclusion in the data [DR14]. Noise, added to the function output, is one way to achieve differential privacy, e.g., via the *Laplace mechanism* [DR14]:

Definition 3 (Laplace mechanism)**.** *Given a function $f : U^n \to \mathbb{R}$ with sensitivity*

$$\max_{\forall D \simeq D'} \big|f(D) - f(D')\big|,$$

privacy parameter ε, and a database D, the Laplace mechanism *releases*

$$f(D) + r,$$

where r is drawn from the Laplace distribution $Lap(b;x)$ with scale $b = \Delta f/\varepsilon$ and density

$$\frac{1}{2b} e^{-|x|/b}.$$

The alternative to additive noise is probabilistic output selection via the *exponential mechanism*, introduced by McSherry and Talwar [MT07]. The exponential mechanism expands the application of differential privacy to functions with non-numerical output, or when the output is not robust to additive noise, e.g., the median function [LLSY16]. The mechanism is exponentially more likely to select "good" results where "good" is quantified via a utility function $u(D,r)$ which takes as input a database $D \in U^n$, and a potential output $r \in \mathscr{R}$ from a fixed set of arbitrary outputs $\mathscr{R}$. Informally, higher utility means the output is more desirable and its selection probability is increased accordingly.

Definition 4 (Exponential Mechanism)**.** *For any utility function $u : (U^n \times \mathscr{R}) \to \mathbb{R}$ and a privacy parameter ε, the exponential mechanism $\mathsf{EM}_u^\varepsilon(D)$ outputs $r \in \mathscr{R}$ with probability proportional to $\exp(\frac{\varepsilon u(D,r)}{2\Delta u})$, where*

$$\Delta u = \max_{\forall r \in \mathscr{R}, D \simeq D'} \left| u(D,r) - u\left(D',r\right) \right|$$

is the sensitivity of the utility function. That is,

$$Pr[\mathsf{EM}_u^\varepsilon(D) = r] = \frac{\exp\left(\frac{\varepsilon u(D,r)}{2\Delta u}\right)}{\sum_{r' \in \mathscr{R}} \exp\left(\frac{\varepsilon u(D,r')}{2\Delta u}\right)}. \tag{2.1}$$

We omit u, ε, D, i.e., write EM, if they can be derived from the context.

DP algorithms $\mathscr{M}$ can be implemented in different models which we describe next.

Why we consider the central model

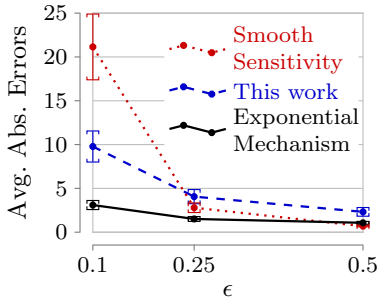
We briefly recall the definition of the various models as detailed in D5.4 [PFOR21]. In the *central model*, the parties send their raw data to a central server – which is trusted and performs the randomization. In the *local model*, the parties locally randomize their data before sending it to an unstrusted server. The *shuffle model* is similar to the local model, however, a trusted shuffler is added between client and server, and the shuffler permutes and forwards the randomized client values. On one hand, the central model enjoys optimal accuracy as the randomization is only applied once, however, it requires a trusted third party. On the other hand, the local model does not require a trusted third party but the accuracy is limited. Accuracy and trust assumptions of the shuffle model lies between the above mentioned models.

Our goal is high accuracy without trusted parties. Therefore, we simulate the trusted third party from the central model via secure multi-party computation – as often used in DP literature [DKM$^+$06, GX17, TKZ16, RN10, PL15, EKM$^+$14].

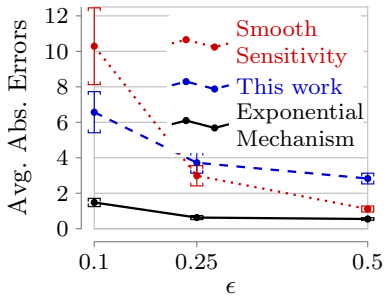
Why we use the exponential mechanism

Next, we illustrate why the exponential mechanism offers better accuracy than additive noise w.r.t. the DP median. Recall, the noise depends on the sensitivity of function f and the privacy parameter ε. The sensitivity is the largest difference a single change in *any possible database* can have on the function result. *Smooth sensitivity*, developed by Nissim et al. [NRS07], additionally analyzes the data to provide instance-specific additive noise that is often much smaller. Formally, given a sorted dataset D with n elements (assumed to be an even number which can be achieved via padding), smooth sensitivity for the median is defined as follows

$$\max_{k=0,..,n} e^{-k\varepsilon} \max_{t=0,..,k+1} \left(d_{\frac{n}{2}+t} - d_{\frac{n}{2}+t-k-1} \right).$$



(a) Credit card transactions [ULB18], first 10^5 payment records in Cents.



(b) Walmart supply chain data [Kag18], 175k shipment weights as integers.

Figure 2.1: Absolute errors, averaged for 100 differentially private median computations via Laplace mechanism with smooth sensitivity, this work, and the exponential mechanism.

However, computation of smooth sensitivity requires access to the entire dataset D, otherwise the error increases further[1], which prohibits efficient (secure) computation with high accuracy. Li et al. [LLSY16] note that the Laplace mechanism is ineffective for the median as (smooth) sensitivity can be high. Additionally, they present a median utility function for the exponential mechanism with low, *data-independent* sensitivity, which we use in our protocol. To illustrate that additive noise can be high, we empirically evaluated the absolute error of the Laplace mechanism with smooth sensitivity, the exponential mechanism, and our protocol in Figure 2.1 on real-world datasets [Kag18, ULB18]. Our protocol uses the exponential mechanism in multiple steps, and while the accuracy is not the same as for (single use of) the exponential mechanism, we do not require a trusted third party. Overall, we achieve better accuracy than additive noise for low ε (corresponding to high privacy protection) with better scalability than the exponential mechanism. We provide our accuracy bounds in Section 2.3.4, further empirical evaluations w.r.t. related work in Section 2.5.3, and describe related work in Section 2.6.

2.2.2 Secure Multi-party Computation

Secure multi-party computation (MPC) allows a set of three or more parties $\mathscr{P} = \{P_1, \ldots, P_n\}$, where party P_i holds sensitive input d_i, to jointly compute a function $y = f(d_1, \ldots, d_n)$ while protecting their inputs [Gol09]. The computation must be *correct*, i.e., the correct y is computed, and *secret*, i.e., only y and nothing else is revealed. There are two main implementation paradigms for MPC [EKR18, KPR18]: *garbled circuits* [Yao86][2], where the parties construct a (large, encrypted) circuit and evaluate it at once, and *secret sharing* [Sha79, BDOZ11, DPSZ12, NNOB12], where the parties interact for each circuit gate. In general, the former allows for constant number of rounds but requires larger bandwidth (as fewer, but bigger messages are sent), and the latter has low bandwidth (small messages per gate) and high throughput, where the number of rounds depends on the circuit depth. We will focus on secret-sharing-based MPC as our goal is an efficient implementation in a network with reasonable latency. Informally, a (t,n)-secret sharing scheme splits a secret s into n shares s_i and at least t shares are required to reconstruct the secret. We use $\langle s \rangle = (s_1, \ldots, s_n)$

[1] Smooth sensitivity approximations exist that provide a factor of 2 approximation in linear-time, or an additive error of $\max(U)/\text{poly}(|D|)$ in sublinear-time [NRS07, Section 3.1.1]. Note that this error e is w.r.t. smooth sensitivity s, the additive noise is even larger with $\text{Laplace}((s+e)/\varepsilon)$.

[2] Yao described a garbled circuit for two parties in an oral presentation about secure function evaluation [Yao86], the first written description is from [GMW87], and the first proof was given in [LP09].

to denote the sharing of s among n parties (for a formal definition see, e.g., Evans et al. [EKR18]). Recent works, e.g., SCALE-MAMBA [AKR+20], BDOZ [BDOZ11], SPDZ [DPSZ12], improve MPC performance by combining a computationally secure *offline phase*, to exchange correlated randomness (e.g., Beaver triples [Bea91]), with an information-theoretic secure *online phase*. The former is generally more efficient since the latter requires asymmetric cryptography [KRSW18]. MPC can be implemented in two models with different trust assumptions: in the *semi-honest model* (passive) adversaries do not deviate from the protocol but gather everything created during the run of the protocol, in the *malicious model* (active) adversaries can deviate from the protocol (e.g., alter messages).

In this work we consider n *input parties* with sensitive input, and m semi-honest *computation parties*, i.e., non-colluding untrusted servers. We evaluated with $3, 6$ and 10 computation parties. The input parties create and send shares of their input to the computation parties, which run the secure computation on their behalf. We assume semi-honest parties but explain how to extend our protocol to malicious parties and implement our protocol with the SCALE-MAMBA framework [AKR+20].

2.3 Secure EM for Median Selection

We implement a multi-party computation of the exponential mechanism EM for rank-based statistics enabling distributed parties to learn the differentially private median of their joint data. There are two challenges for multi-party computation of the exponential mechanism:

(i) the running time complexity is linear in the size of the data universe, $|U|$, as selection probabilities for *all* possible outputs in U are computed,

(ii) the general mechanism is too inefficient for general secure computation as selection probability computation requires $|U|$ exponentiations over floating-point numbers.

We solve these challenges by (i) recursively dividing the data universe into subranges to achieve sublinear running time in $|U|$, and (ii) focusing on utility functions which allow efficient selection probability computation. We call such utility functions *decomposable*, which we formalize in Section 2.3.1, and give example applications.

In the following, we describe an overview of our solution. Note that we use $\lceil\cdot\rceil$ (resp. $\lfloor\cdot\rfloor$) to indicate rounding up (resp. down) to the nearest integer. We efficiently compute the exponential mechanism with running time complexity sublinear in the size of the data universe U by dividing U into k subranges. We select the best subrange and also split it into k subranges for the next iteration, until the last subrange is small enough to directly select the final output from it. After $\lceil \log_k |U| \rceil$ iterations the selected subrange contains only one element. Each subrange selection increases the overall privacy loss ε, and we enable users to select a trade-off between running time, privacy loss and accuracy by presenting three protocols to compute unnormalized selection probabilities, which we call *weights*, w.r.t. ε:

- $\mathsf{Weights}^{\ln(2)}$ fixes $\varepsilon = \ln(2)$ to compute $\exp(\varepsilon y)$ as 2^y,
- $\mathsf{Weights}^{\ln(2)/2^d}$ allows $\varepsilon = \frac{\ln(2)}{2^d}$ for some integer $d > 0$,
- $\mathsf{Weights}^{*}$ supports arbitrary ε.

On a high-level, we have three phases in each iteration:

Application	*Utility*
Convex optimization: find x that minimizes $\sum_{i=1}^{n} l(x,d_i)$ with convex loss function l defined over D; e.g., empirical risk minimization in machine learning [BST14, STU17], and integer partitions (password frequency lists) [BDB16]	$-\sum_{i=1}^{n} l(x,d_i)$
Unlimited supply auction: find price x maximizing revenue $x\sum_i b_i(x)$, where bidder demand curve b_i indicates how many goods bidder i will buy at price x; e.g., digital goods [MT07]	$x\sum_i b_i(x)$
Frequency: select x based on its frequency in D; e.g., mode [LLSY16]	$\sum_{i=1}^{n} \mathbb{1}_{x=d_i}$
Rank-based statistics: select x based on its rank in sorted D; e.g., k^{th}-ranked element [LLSY16]	See Section 2.3.2

Table 2.1: Applications with *decomposable* utility functions.

1. *Evaluate*: Each party locally computes the basis for utility scores for each subrange.
2. *Combine*: They combine their results into a global result and compute selection probabilities.
3. *Select*: Finally, they select an output based on its selection probabilities.

The results of the evaluation step are computed over sensitive data and might also be sensitive (e.g., utility functions for median and mode leak exact counts [LLSY16]). Therefore, we combine them via MPC to preserve privacy. To ensure efficient implementation of the combination step we require utility functions to have a certain structure as detailed next.

2.3.1 Decomposability & Applications

Recall, each party P_i holds a single value d_i (we can generalize to datasets D_i). To combine local utility scores per party into a global score for all, we require utility functions to be *decomposable*:

Definition 5 (Decomposability)**.** *We call a function* $u : (U^n \times \mathscr{R}) \to \mathbb{R}$ decomposable *w.r.t. function* $u' : (U \times \mathscr{R}) \to \mathbb{R}$ *if* $u(D,x) = \sum_{i=1}^{n} u'(d_i,x)$ *for* $x \in \mathscr{R}$ *and* $D = \{d_1,\ldots,d_n\}$.

We use decomposability to easily combine utility scores in $\mathsf{Weights}^{\ln(2)}$, $\mathsf{Weights}^{\ln(2)/2^d}$, and to avoid secure evaluation of the exponential function in $\mathsf{Weights}^{*}$[3]. If u is decomposable, users can compute weights locally, and securely combine them via multiplications:

$$\prod_i \exp(u'(d_i,x)\varepsilon) = \exp(\sum_i u'(d_i,x)\varepsilon) = \exp(u(D,x)\varepsilon).$$

Decomposability is satisfied by a wide range of selection problems. Counts are clearly decomposable and so are utility functions that can be expressed as a sum of utility scores. Examples for decomposable utility functions are listed in Table 2.1. An additional example for decomposable utility is gradient compressed federated learning (to solve non-convex optimization problems with

[3] Secure exponentiation is complex [DFK+06, AS19, Kam15, ABZS13], requiring many interactive rounds, and we want to avoid the expensive computational overhead.

efficient communication), e.g., signSGD [BWAA18]: each party only provides the sign of the gradient (as local utility score) and the aggregate of all signs is used to perform the update step.

To be sublinear in the size of the universe we consider decomposability w.r.t. ranges instead of elements: parties only report one utility score per range, instead of one score per element. Decomposability for elements $x \in U$ does not imply decomposability for ranges $R \subset U$[4]. However, we present a decomposable utility function w.r.t. ranges for rank-based statistics next.

2.3.2 Decomposable Median Utility Function

First, we describe the median utility function [LLSY16]. Then, we present a reformulation more convenient for secure implementation and show that it is decomposable.

Li et al. [LLSY16, Section 2.4.3] quantify an element's utility via its *rank* relative to the median. The rank of $x \in U$ in a dataset D is the number of values in D smaller than x. More formally, $\text{rank}_D(x) = |\{d \mid d \in D : d < x\}|$. Note that for the median we have $\mathscr{R} = U$, which means every universe element is a potential output. As U can be large, we divide U in k equal-sized ranges, and define utility per range next.

Definition 6 (Median utility function)**.** *The* median utility function $u_\mu : (U^n \times U) \to \mathbb{Z}$ *gives a utility score for a range* $R = [r_l, r_u)$ *where* $r_l, r_u \in U$ *w.r.t.* $D \in U^n$ *as*

$$u_\mu(D,R) = -\min_{\text{rank}_D(r_l) \le j \le \text{rank}_D(r_u)} \left| j - \frac{n}{2} \right|.$$

We focus on MPC of the differentially private median with rank $n/2$ but Definition 6 supports any k^{th}-ranked element. The sensitivity of u_μ is $1/2$ since adding an element increases $n/2$ by $1/2$ and j either increases by 1 or remains the same [LLSY16]. Thus, the denominator $2\Delta u$ in the exponents of (2.1) equals 1, and we will omit it in the rest of this work.

To compute u_μ one needs to find rank j minimizing the distance between the median and all range elements by iterating over all j where $\text{rank}_D(r_l) \le j \le \text{rank}_D(r_u)$. However, a naive implementation of u_μ leaks information as the iteration count depends on the number of duplicates in the data. We adapt u_μ next to remove this leakage. To avoid iterating over range elements observe that the utility for a range $R = [r_l, r_u)$ is defined by the element in the range closest to the median μ. Thus, it suffices to consider three cases: The range is either positioned "before" the median ($r_u \le \mu$), contains it, or comes "after" it ($r_l > \mu$). This observation leads us to the following definition without iterations:

Definition 7 (Simplified median utility function)**.** *The* median utility function $u^c_\mu : (U^n \times U) \to \mathbb{Z}$ *gives a utility score for a range* $R = [r_l, r_u)$ *of* U *w.r.t.* $D \in U^n$ *as*

$$u^c_\mu(D,R) = \begin{cases} \text{rank}_D(r_u) - \frac{n}{2} & \text{if } \text{rank}_D(r_u) < \frac{n}{2} \\ \frac{n}{2} - \text{rank}_D(r_l) & \text{if } \text{rank}_D(r_l) > \frac{n}{2} \\ 0 & \text{else} \end{cases}.$$

In the following, we generalize from a single value per (input) party, d_i, to multiple values, i.e., dataset D_i, as computation parties operate on datasets later on. Definition 6 and 7 are equivalent

[4] Consider the mode, i.e., the most frequent element. E.g., for two parties with datasets $D_1 = \{1,1,1,2,2\}, D_2 = \{2,2,3,3,3\}$ the mode per dataset is 1 resp. 3 but the mode for the combined data is 2.

1. Set $s = \lceil \log_k |U| \rceil$ and split privacy budget ε into $\varepsilon_1, \ldots, \varepsilon_s$
2. Initialize $S = U$ and repeat below steps s times:
   (a) Every party $p \in \mathscr{P}$ divides S into k equal-sized subranges $\{R^i = [r_l^i, r_u^i)\}_{i=1}^k$
      i. if $\varepsilon_j = \ln(2)/2^d$ in step j (with integer $d \geq 0$), input $\left\{\text{rank}_{D_p}(r_l^i), \text{rank}_{D_p}(r_u^i)\right\}_{i=1}^k, d$
      ii. else input $\left\{e^{\varepsilon\left(\text{rank}_{D_p}(U[r_u^i]) - |D_p|/2\right)}, e^{\varepsilon\left(|D_p|/2 - \text{rank}_{D_p}(U[r_l^i])\right)}\right\}_{i=1}^k, \varepsilon_j$
   (b) The functionality combines the inputs (Section 2.3.2) and outputs $S = R^i$ with probability proportional to $\exp(u_\mu^c(D, R^i)\varepsilon_j)$

Figure 2.2: Ideal functionality $\mathscr{F}_{\mathsf{EM}^*}$ for EM^*.

as can be seen by proof by cases and u_μ^c is decomposable w.r.t.:

$$u'(D_i, R) = \begin{cases} \text{rank}_{D_i}(r_u) - \frac{|D_i|}{2} & \text{if } \text{rank}_D(r_u) < \frac{n}{2} \\ \frac{|D_i|}{2} - \text{rank}_{D_i}(r_l) & \text{if } \text{rank}_D(r_l) > \frac{n}{2} \\ 0 & \text{else} \end{cases},$$

where $\text{rank}_D(r) = \sum_{i=1}^n \text{rank}_{D_i}(r)$ for range endpoints r. We will use both utility definitions interchangeably. Specifically, we use u_μ to simplify notation in our accuracy proofs (Section 2.3.4), and u_μ^c in our implementation (Section 2.4).

For implementations $\mathsf{Weights}^{\ln(2)}$, $\mathsf{Weights}^{\ln(2)/2^d}$ the parties input *ranks* for lower and upper range endpoints (as in u' above), which we combine (as u_μ^c) to efficiently compute weights. For $\mathsf{Weights}^*$ we let the parties input *weights*, i.e., $\exp(\varepsilon u')$, which we can efficiently combine via multiplication. In more detail, weights for u' are:

$$e^{\varepsilon \cdot u'(D_i, R)} = \begin{cases} e^{\varepsilon\left(\text{rank}_{D_i}(r_u) - \frac{|D_i|}{2}\right)} & \text{if } e^{\varepsilon\left(\text{rank}_D(r_u) - \frac{n}{2}\right)} < 1 \\ e^{\varepsilon\left(\frac{|D_i|}{2} - \text{rank}_{D_i}(r_l)\right)} & \text{if } 1 > e^{\varepsilon\left(\frac{n}{2} - \text{rank}_D(r_l)\right)} \\ 1 & \text{else} \end{cases},$$

where, e.g., $e^{\varepsilon\left(\text{rank}_D(r) - \frac{n}{2}\right)} = \prod_{i=1}^n e^{\varepsilon\left(\text{rank}_{D_i}(r) - \frac{|D_i|}{2}\right)}$ for range endpoints r. Given these inputs, we are ready to describe an idealized version of our protocol next.

2.3.3 Ideal Functionality $\mathscr{F}_{\mathsf{EM}^*}$

The ideal functionality $\mathscr{F}_{\mathsf{EM}^*}$ in Figure 2.2 describes our DP median protocol EM^* as executed by a trusted third party, which we later replace by implementing $\mathscr{F}_{\mathsf{EM}^*}$ with MPC. We iteratively select subranges of universe U w.r.t. DP median via the exponential mechanism. After $s = \lceil \log_k |U| \rceil$ steps the last selected subrange contains only the DP median. We split ε, also called *privacy budget*, into s parts such that $\varepsilon = \sum_{j=1}^s \varepsilon_j$, and consume ε_j for each subrange selection. (We describe the budget composition in Section 2.3.4 and provide a heuristic in Section 2.5.) Overall, $\mathscr{F}_{\mathsf{EM}^*}$ provides ε-differential privacy:

Theorem 1. *$\mathscr{F}_{\mathsf{EM}^*}$, with privacy parameter ε_j in step $j \in \{1, \ldots, s\}$, is ε-differentially private for $\varepsilon = \sum_{j=1}^s \varepsilon_j$.*

Proof. $\mathscr{F}_{\mathsf{EM}^*}$ performs s sequential steps, and each step applies the exponential mechanism $\mathsf{EM}^{\varepsilon_i}_{u^c_\mu}$. Since $\mathsf{EM}^{\varepsilon_i}_{u^c_\mu}$ is $(2\varepsilon_i\Delta u^c_\mu)$-DP [MT07], with sensitivity $\Delta u^c_\mu = 1/2$ [LLSY16], we have ε_i-DP per step. Thus, according to the composition theorem [DR14], the total privacy budget after all steps is $\sum_{j=1}^{s}\varepsilon_j$. □

2.3.4 Accuracy of Differentially Private Median

We express *accuracy* as the absolute error between differentially private and actual median. In more detail, the absolute error is bounded by α with probability at least $1-\beta$, known as (α,β)-accuracy. In the following, we discuss how the data distribution influences accuracy. Then, we present worst-case bounds on the accuracy of the exponential mechanism for median selection.

Data Distribution

Accuracy depends on the data distribution, specifically, on *gaps* $d_{i+1}-d_i$, and *duplicates* $d_i = d_j$ with $i \neq j$[5]. Recall, a DP mechanism bounds the probability that dataset D and its neighbor D' can be distinguished from the mechanism output. As neighbor D' may contain values from the gaps of D, these gap values must be output with a non-zero probability. As a worst-case example, consider a dataset with universe $U = \{0,1,\ldots,10^9\}$, and equal number of duplicates for 0 and 10^9. Then, smooth sensitivity is extremely large with 10^9 and the exponential mechanism outputs a value at uniform random. However, for such pathological, worst-case data even the actual median does not provide much insight. On the other hand, the number of duplicates in the data can increase accuracy dramatically. For example, consider a dataset where the median has $2c$ duplicates: $d_{n/2\pm i} = d_{n/2}$ for $i \in \{1,\ldots,c\}$. Then, the probability that the exponential mechanism outputs the median is $\exp(c\varepsilon)$ times higher than for any other element. Such duplicates also fit the intuition that the median is a "typical" value from the data that represents it well. In general, the probability to output a "bad" element x decreases exponentially in $\sum c_i$, where $c_i \geq 1$ are duplicate counts of "good" elements y_i, which are closer to the median than x.

Accuracy Bounds

In the following, we show that the output of $\mathsf{EM}^{\varepsilon}_{u}(D,\mathscr{R})$ contains an element at most $\left\lfloor \frac{\ln(|\mathscr{R}|/\beta)}{\varepsilon} \right\rfloor$ positions away from the median in the sorted data. Note that $|\mathscr{R}|$ is k if we select among k subranges or $|U|$ if we output elements directly.

For our accuracy proofs we structure the universe as a tree: we set U as the root of a tree of height $\log_b|U|$, for some base b, with k child nodes per parent. The child nodes are equal-sized subranges of the parent node and R^j_i denotes the i^{th} subrange in level j.

Theorem 2 (Median Accuracy for Ranges)**.** *Fixing a database D of size n with a set of k subranges $\mathscr{R} = \{R^j_1,\ldots,R^j_k\}$ of data universe U. Then, output of $\mathsf{EM}^{\varepsilon}_{u}(D,\mathscr{R})$ contains an element at most $\left\lfloor \frac{\ln(k/\beta)}{\varepsilon} \right\rfloor$ positions away from median position $\frac{n}{2}$ with probability at least $1-\beta$.*

Our proof uses Corollary 3.12 from [DR14], which we restate as the following Lemma:

[5] To simplify the explanation, assume the universe to consists of consecutive integers, i.e., $U = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$ with $a,b \in \mathbb{Z}$.

Lemma 1 (Accuracy of the Exponential Mechanism)**.** *Fixing a database D, and let $OPT = \max_{r\in\mathscr{R}} u(D,r)$ denote the maximum utility score of any element $r \in \mathscr{R}$, we have*

$$Pr\left[u(D,\mathsf{EM}_u^\varepsilon(D,\mathscr{R})) \le OPT - \frac{2\Delta u}{\varepsilon}(\ln|\mathscr{R}|+t)\right] \le \exp(-t).$$

Proof of Theorem 2. First, we bound the utility difference between optimal and selected output. Then, we translate this to a bound on the output's rank.

The complementary of Lemma 1 with $\Delta u = \frac{1}{2}$ is

$$\Pr\left[\mathrm{OPT} - u(D,\mathsf{EM}_u^\varepsilon(D,\mathscr{R})) < \frac{\ln|\mathscr{R}|+t}{\varepsilon}\right] > 1-\exp(-t).$$

Let $R_i^j = [r_l, r_u)$ be the output of $\mathsf{EM}_u^\varepsilon(D,\mathscr{R})$. Recall, that for median utility $\mathrm{OPT} = 0$, then,

$$\begin{aligned}\mathrm{OPT} - u(D,\mathsf{EM}_u^\varepsilon(D,\mathscr{R})) &= 0 - u(D,R_i^j)\\ &= \min_{\mathrm{rank}_D(r_l)\le j\le \mathrm{rank}_D(r_u)}\left|j-\frac{n}{2}\right|.\end{aligned}$$

Next, we consider different cases for R_i^j to bound the rank difference between the selected range and the range that contains the median. Assume median $\mu \notin R_i^j$, as otherwise the bound holds trivially, and let d denote the utility difference $\mathrm{OPT} - u(D,\mathsf{EM}_u^\varepsilon(D,\mathscr{R}))$.

For $r_u < \mu$ we have $d = |\mathrm{rank}_D(r_u) - \frac{n}{2}| = \frac{n}{2} - \mathrm{rank}_D(r_u)$ from which we obtain $\mathrm{rank}_D(r_u) > \frac{n}{2} - \frac{\ln|\mathscr{R}|+t}{\varepsilon}$ with probability at least $1-\exp(-t)$. Analog, for $r_l > \mu$ we have $d = \mathrm{rank}_D(r_l) - \frac{n}{2}$, and obtain $\mathrm{rank}_D(r_l) < \frac{n}{2} + \frac{\ln|\mathscr{R}|+t}{\varepsilon}$ with the same probability. Altogether, R_i^j is at most $\left\lfloor\frac{\ln|\mathscr{R}|+t}{\varepsilon}\right\rfloor$ rank positions away from median rank $n/2$ with probability at least $1-\exp(-t)$. We have $k = |\mathscr{R}|$ and setting $\beta = \exp(-t)$ concludes the proof. □

To obtain an absolute error with regards to data elements, consider universe elements instead of subranges as the output of the exponential mechanism.

Corollary 1 (Median Accuracy)**.** *Fixing a sorted database D of size n, let μ be the median of D, and $\widehat{\mu}$ the output of $\mathsf{EM}_u^\varepsilon(D,U)$. Then, absolute error $|\mu-\widehat{\mu}|$ is at most*

$$\max_{i\in\{+1,-1\}\cdot\left\lfloor\frac{\ln(|U|/\beta)}{\varepsilon}\right\rfloor}\left|d_{\frac{n}{2}+i} - d_{\frac{n}{2}}\right|$$

with probability at least $1-\beta$.

The proof follows directly from Theorem 2 with $|\mathscr{R}| = |U|$.

Note that it is more likely to select a "good" subrange as it is to directly select a "good" element from the entire universe (as $k \ll |U|$). However, sequential (subrange) selections consumes ε_j per selection step j which adds up to a total privacy budget of $\varepsilon = \sum_j \varepsilon_j$ as described in Section 2.3.3. We now show how to choose ε_j to select the subrange containing the median in each iteration step with probability at least $1-\beta$.

Theorem 3 (Choice of ε)**.** *Let $\mathscr{R} = \{R_1^j,\ldots,R_k^j\}$, where $R_i^j = [r_l, r_u)$ contains the median, and $n_{ij} = \min\{|\mathrm{rank}_D(\mu) - \mathrm{rank}_D(r_l)|, |\mathrm{rank}_D(r_u+1) - \mathrm{rank}_D(\mu+1)|\}$ is the minimum count of data elements in R_i^j smaller resp. larger than the median. Then, $\mathsf{EM}_u^\varepsilon(D,\mathscr{R})$ selects R_i^j with probability at least $1-\beta$ if*

$$\varepsilon_j \ge \frac{\ln(k/\beta)}{n_{ij}}.$$

Proof. Ranges R_h^j without the median have a rank at least n_{ij} positions away from median rank. More formally,

$$\text{OPT} - u(D, R_h^j) \geq \left| \left(\frac{n}{2} \pm n_{ij} \right) - \frac{n}{2} \right| = n_{ij}.$$

According to Lemma 1 we have $\Pr\left[n_{ij} \geq \frac{\ln|\mathscr{R}|+t}{\varepsilon_j} \right] \leq \exp(-t)$. Thus, for $\varepsilon_j \geq \frac{\ln|\mathscr{R}|+t}{n_{ij}}$ the probability that any range R_h^j is selected is at most $\exp(-t)$. We have $k = |\mathscr{R}|$ and setting $\beta = \exp(-t)$ concludes the proof. □

Parameter ε_j is undefined for $n_{ij} = 0$, i.e., when the median is a range endpoint[6]. Note that the exact value of n_{ij} is data-dependent. E.g., for the uniform distribution $n_{ij} \approx |D|/k^j$. A differentially private n_{ij} can be efficiently computed by distributed sum protocols [DKM+06, GX17, TKZ16, RN10] as it is just a count of data elements. However, a differentially private count also consumes a portion of the privacy parameter. For low epsilon (e.g., $\varepsilon = 0.1$) we want to use the entire privacy budget on the actual median selection to achieve high accuracy. Thus, we use a heuristic in our evaluation: larger subranges, that hold exponentially more elements, receive exponentially smaller portions ε_j of the privacy budget (see Section 2.5 for details).

2.4 MPC for Differentially Private Median

In the following, we describe details of our protocol EM^*, which implements ideal functionality $\mathscr{F}_{\mathsf{EM}^*}$, analyse its running time and security.

On a high-level, our protocol recursively selects the best subrange until the DP median is found: First, each party locally *evaluates* a utility score (or weight) for each subrange. They *combine* their results into a global result. Then, they *select* a subrange based on the combined result. We use upper case letters to denote arrays in our protocol, and $A[j]$ denotes the j^{th} element in array A. Our protocol uses integers as well as floating point numbers. We adopt the notation from Aliasgari et al. [ABZS13] and represent a floating-point number f as $(1-2s)(1-z)\cdot v \cdot 2^x$ with sign bit s set when the value is negative, zero bit z only set when the value is zero, l_v-bit significand v, and l_x-bit exponent x. The sharing of a floating point value f is a 4-tuple $(\langle v\rangle, \langle x\rangle, \langle s\rangle, \langle z\rangle)$, which we abbreviate as $\langle f\rangle_{\mathsf{FL}}$. To refer to, e.g., the significand v of f we will write $f.v$. The basic MPC protocols used in our protocol are listed in Table 2.2. We prefix MPC protocols for integers with Int and floating point versions with FL.

2.4.1 Subrange Selection

On a high level, protocol EM^*, implemented in Algorithm 1, computes selection weights for possible outputs (via Algorithm 2) and selects an output according to these weights (via Algorithm 3 or 4). We assume that the universe U and combined data size n are known to all parties (note that the latter can be hidden via padding [AMP10]). Recall, that efficient weight computation and selection are the main challenges for our secure exponential mechanism. Straightforward selection over all universe elements is linear in the size of U. To achieve a running time sublinear in the size of U we select subranges instead: Algorithm 1 selects one of k subranges based on their median utility. The selected subrange is recursively divided into k subranges until the last

[6] An undefined ε_j can be avoided by using an additional discretization of the universe, with different subrange endpoints, and switching to it if a (differentially private) check suggests $n_{ij} = 0$ [DL09].

MPC protocol	*Output / Functionality*
$\mathsf{Rec}(\langle a\rangle)$	a, reconstructed from $\langle a\rangle$
$\mathsf{Add}(\langle a\rangle,\langle b\rangle)$	$\langle a+b\rangle$
$\mathsf{Sub}(\langle a\rangle,\langle b\rangle)$	$\langle a-b\rangle$
$\mathsf{Mul}(\langle a\rangle,\langle b\rangle)$	$\langle a\cdot b\rangle$
$\mathsf{Mod2m}(\langle a\rangle,b)$	$\langle a \mod 2^b\rangle$, where b is public
$\mathsf{Trunc}(\langle a\rangle,b)$	$\langle \lfloor a/2^b \rfloor\rangle$, where b is public
$\mathsf{Rand}(b)$	$\langle r\rangle$ with uniform random b-bit value r
$\mathsf{Choose}(\langle a\rangle,\langle b\rangle,\langle c\rangle)$	$\langle a\rangle$ if bit $c=1$ otherwise $\langle b\rangle$
$\mathsf{LT}(\langle a\rangle,\langle b\rangle)$	$\langle 1\rangle$ if $a<b$ else $\langle 0\rangle$
$\mathsf{Int2FL}(\langle a\rangle)$	converts integer a to secret shared float

Table 2.2: Basic MPC protocols [ABZS13, AKR+20] used in EM^*. We prefix protocols for integers with Int and floats with FL.

Algorithm 1 Algorithm EM^*.

Input: Number of subranges k, size n of combined data D, number of selection steps $s \in [1, \lceil \log_k |U| \rceil]$, and $(\varepsilon_1, \ldots, \varepsilon_s)$. Data universe U is known to all parties.
Output: Differentially private median of D.

1: $r_l, r_u \leftarrow 0, |U|$
2: **for** $j \leftarrow 1$ **to** s **do**
3: $\quad r_\# \leftarrow \max\{1, \lfloor \frac{r_u - r_l}{k} \rfloor\}$
4: $\quad k \leftarrow \min\{k, r_u - r_l\}$
5: $\quad$ Define array W of size k
6: $\quad$ **if** $\varepsilon_j = \ln(2)/2^d$ for some integer d **then**
7: $\quad\quad \langle W\rangle_{\mathsf{FL}} \leftarrow \mathsf{Weights}^{\ln(2)/2^d}(r_l, r_u, r_\#, k, n, d)$ `//Alg. 3`
8: $\quad$ **else**
9: $\quad\quad \langle W\rangle_{\mathsf{FL}} \leftarrow \mathsf{Weights}^*(r_l, r_u, r_\#, k, n, \varepsilon_j)$ `//Algorithm 4`
10: $\quad$ **end if**
11: $\quad i \leftarrow \mathsf{Select}(\langle W\rangle_{\mathsf{FL}})$ `//Algorithm 2`
12: $\quad r_l \leftarrow r_l + (i-1)\cdot r_\#$
13: $\quad r_u \leftarrow r_l + r_\#$ **if** $i < k$
14: **end for**
15: **return** Uniform random element in $[U[r_l], U[r_u])$

subrange, after at most $\lceil \log_k |U| \rceil$ iterations, contains only one element: the differentially private median[7]. Alternatively, one can use fewer selection steps s and select an element from the last subrange at uniform random (line 15 in Algorithm 1). We discuss the running time vs. accuracy trade-offs of reduced selection steps in Section 2.5. We implement selection with *inverse transform sampling* (ITS) via binary search in Algorithm 2 similar to [EKM+14]. ITS uses the uniform distribution to realize any distribution based on its cummulative distribution function. Formally, one draws $r \in (0,1]$ at uniform random and outputs the first $R_j \in \mathscr{R}$ with $\sum_{i=1}^{j-1} \Pr[\mathsf{EM}_u^\varepsilon(D,\mathscr{R}) = R_i] \le r < \sum_{i=1}^{j} \Pr[\mathsf{EM}_u^\varepsilon(D,\mathscr{R}) = R_i]$. Recall, we compute unnormalized probabilities (weights), which do not require division for normalization, thus, reducing computation complexity. To use weights instead of probabilities in ITS we only need to multiply r with normalization $N = \sum_{o\in\mathscr{R}} \exp(u(D,o)\varepsilon)$.

[7] To simplify presentation, assume that $\log_k |U|$ is an integer. Otherwise the last subrange might contain less than k elements, and fewer weight computations are needed in the last step.

Algorithm 2 Algorithm Select.

Input: List $\langle W \rangle_{\mathsf{FL}}$ of weights with size k.
Output: Index $j \in [1,k]$ sampled according to $\langle W \rangle_{\mathsf{FL}}$.

1: Define array M of size k `//Probability mass`
2: $\langle M[1] \rangle_{\mathsf{FL}} \leftarrow \langle W[1] \rangle_{\mathsf{FL}}$
3: **for** $j \leftarrow 2$ **to** k **do**
4: $\quad \langle M[j] \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLAdd}(\langle W[j] \rangle_{\mathsf{FL}}, \langle M[j-1] \rangle_{\mathsf{FL}})$
5: **end for**
6: $\langle t \rangle \leftarrow \mathsf{IntRand}(b)$ `//Bitlength b`
7: $\langle f \rangle_{\mathsf{FL}} \leftarrow \mathsf{Int2FL}(\langle t \rangle)$
8: $\langle x \rangle \leftarrow \mathsf{IntSub}(\langle f.x \rangle, \langle b \rangle)$
9: $\langle f \rangle_{\mathsf{FL}} \leftarrow (\langle f.v \rangle, \langle x \rangle, \langle f.z \rangle, \langle f.s \rangle)$
10: $\langle r \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle M[k] \rangle_{\mathsf{FL}}, \langle f \rangle_{\mathsf{FL}})$
11: $i_l \leftarrow 1; i_u \leftarrow k$
12: **while** $i_l < i_u$ **do**
13: $\quad i_m \leftarrow \left\lfloor \frac{i_l + i_u}{2} \right\rfloor$
14: $\quad \langle c \rangle \leftarrow \mathsf{FLLT}(\langle M[i_m] \rangle_{\mathsf{FL}}, \langle r \rangle_{\mathsf{FL}})$
15: $\quad c \leftarrow \mathsf{Rec}(\langle c \rangle)$
16: $\quad i_l \leftarrow i_m + 1$ **if** $c = 1$ **else** $i_u \leftarrow i_m$
17: **end while**
18: **return** i_l

We use decomposable utility functions to combine local evaluations over each party's data into a global utility score for the joint data. Next, we present three solutions to efficiently compute weights for decomposable utility functions.

2.4.2 $\mathsf{Weights}^{\ln(2)}$

We implement $\mathsf{Weights}^{\ln(2)}$ as a special case of our approach $\mathsf{Weights}^{\ln(2)/2^d}$ in Algorithm 3 (with $d = 0$ in line 16). Here, parties locally compute *ranks* which are combined into global utility scores. Weights for these scores use a fixed ε of $\ln(2)$ to let us compute 2^u instead of $\exp(\varepsilon \cdot u)$. Solutions for secure exponentiation of 2^u exist where u is an integer or a float [DFK+06, AS19, Kam15, ABZS13]. When u is an integer (resp. a float) the result 2^u is an integer (resp. float) as well. The complexity of the integer-based solution is linear in the bit-length of u, however, this is not sufficient for us. Recall, that the utility is based on ranks, i.e., counts of data elements, thus u can be roughly as large as the size of the data. An integer representation of 2^u has bit-length u, which is potentially unbounded. Eigner et al. [EKM+14] use the float-based solution from [ABZS13] but we present a more efficient computation in the following. Although our exponent u is an integer, we do not require the result to be an integer as well. We use the representation of floating point numbers as a 4-tuple to construct a new float to represent 2^u as $(2,u,0,0)$, where sign and zero bit are unset, as 2^u cannot be negative or zero. Note that we require no interaction as each party can construct such a float with their share of u. Also, a naive approach requires $2k$ total inputs per party (one per endpoint per k ranges). However, with half-open ranges $[r_l^i, r_u^i)$ in each step i, they overlap for $i > 1$: $r_u^{i-1} = r_l^i$. Thus, the parties only input $k+1$ ranks (Algorithm 3 lines 5, 7).

Algorithm 3 Algorithm $\mathsf{Weights}^{\ln(2)/2^d}$.

Input: Range $[r_l, r_u)$, subrange size $r_\#$, number k of subranges, data size n, and parameter $d \in \{0,1\}$. Subrange ranks $\mathrm{rank}_{D_p}(\cdot)$ are input by each party $p \in \{1,\ldots,m\}$.

Output: List of weights.

1: Define arrays R of size $k+1$, W of size k; initialize R with zeros
2: **for** $p \leftarrow 1$ **to** m **do** `//Get input from each party`
3: **for** $j \leftarrow 1$ **to** k **do** `//Divide range into k subranges`
4: $i_l \leftarrow r_l + (j-1) \cdot r_\#$
5: $\langle R[j] \rangle \leftarrow \mathsf{IntAdd}(\langle R[j] \rangle, \langle \mathrm{rank}_{D_p}(U[i_l]) \rangle)$
6: **end for**
7: $\langle R[k+1] \rangle \leftarrow \mathsf{IntAdd}(\langle R[k+1] \rangle, \langle \mathrm{rank}_{D_p}(U[r_u]) \rangle)$
8: **end for**
9: **for** $j \leftarrow 1$ **to** k **do**
10: $\langle u_u \rangle \leftarrow \mathsf{IntSub}(\langle R[j+1] \rangle, \langle \frac{n}{2} \rangle)$
11: $\langle u_l \rangle \leftarrow \mathsf{IntSub}(\langle \frac{n}{2} \rangle, \langle R[j] \rangle)$
12: $\langle c_u \rangle \leftarrow \mathsf{IntLT}(\langle R[j+1] \rangle, \langle \frac{n}{2} \rangle)$
13: $\langle c_l \rangle \leftarrow \mathsf{IntLT}(\langle \frac{n}{2} \rangle, \langle R[j] \rangle)$
14: $\langle t \rangle \leftarrow \mathsf{IntChoose}(\langle u_u \rangle, \langle 0 \rangle, \langle c_u \rangle)$
15: $\langle u \rangle \leftarrow \mathsf{IntChoose}(\langle u_l \rangle, \langle t \rangle, \langle c_l \rangle)$
16: **if** $d = 0$ **then**
17: $\langle W[j] \rangle_{\mathsf{FL}} \leftarrow (\langle 2 \rangle, \langle u \rangle, \langle 0 \rangle, \langle 0 \rangle)$ `//float` $\langle 2^u \rangle$
18: **else**
19: $\langle t \rangle \leftarrow \mathsf{IntTrunc}(\langle u \rangle, d)$
20: $\langle e \rangle_{\mathsf{FL}} \leftarrow (\langle 2 \rangle, \langle t \rangle, \langle 0 \rangle, \langle 0 \rangle)$
21: $\langle c \rangle \leftarrow \mathsf{IntMod2m}(\langle u \rangle, d)$
22: $\langle s \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLChoose}(\langle 1 \rangle_{\mathsf{FL}}, \langle \sqrt{2} \rangle_{\mathsf{FL}}, \langle c \rangle)$
23: $\langle W[j] \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle e \rangle_{\mathsf{FL}}, \langle s \rangle_{\mathsf{FL}})$
24: **end if**
25: **end for**
26: **return** $\langle W \rangle_{\mathsf{FL}}$

2.4.3 $\mathsf{Weights}^{\ln(2)/2^d}$

Next, we generalize the weight computation to support $\varepsilon = \ln(2)/2^d$ for integers $d \geq 1$. To illustrate our approach, we implement $\mathsf{Weights}^{\ln(2)/2^d}$ in Algorithm 3 for $d = 1$, and describe the approach for any integer d: Recall, our goal is to compute the weight $\exp(\varepsilon u)$ with efficient MPC protocols. As we can efficiently compute $2^{\varepsilon u}$ if εu is an integer, we approximate the weight by truncating εu to an integer before exponentiation with base 2. To avoid a loss of precision we correct this approximation with a multiplicative term based on the truncated remainder. More formally, with ε as above the weight for u is

$$2^{u/2^d} = 2^{\lfloor u/2^d \rfloor} \cdot 2^{(u \mod 2^d)/2^d}.$$

First, we compute $2^{\lfloor u/2^d \rfloor}$ (lines 19–21 in Algorithm 4). Then, we multiply this with one of 2^d constants of the form $2^{(u \mod 2^d)/2^d}$. E.g., for $d = 1$, we either use 1, if u is even, or $\sqrt{2}$ otherwise (line 22). The constants themselves are not secret and can be pre-computed. Which constant was selected, leaks the last d bits from u, thus, we choose them securely.

Algorithm 4 Algorithm Weights*.

Input: Range $[r_l, r_u)$, subrange size $r_\#$, number k of subranges, data size n, and ε. Subrange weights $e^{\varepsilon(\cdot)}$ are input by each party $p \in \{1,\dots,m\}$.
Output: List of weights.

1: Define arrays W^l, W^u, W of size k; initialize W^l, W^u with ones
2: **for** $p \leftarrow 1$ **to** m **do** `//Get input from each party`
3: **for** $j \leftarrow 1$ **to** k **do** `//Divide range into k subranges`
4: $i_l \leftarrow r_l + (j-1)\cdot r_\#$
5: $i_u \leftarrow r_u$ **if** $j = k$ **else** $r_l + j \cdot r_\#$
6: $\langle W^l[j]\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle W^l[j]\rangle_{\mathsf{FL}}, \langle e^{\varepsilon\left(\frac{|D_p|}{2} - \mathrm{rank}_{D_p}(U[i_l])\right)}\rangle_{\mathsf{FL}})$
7: $\langle W^u[j]\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle W^u[j]\rangle_{\mathsf{FL}}, \langle e^{\varepsilon\left(\mathrm{rank}_{D_p}(U[i_u]) - \frac{|D_p|}{2}|\right)}\rangle_{\mathsf{FL}})$
8: **end for**
9: **end for**
10: **for** $j \leftarrow 1$ **to** k **do**
11: $\langle c_u\rangle \leftarrow \mathsf{FLLT}(\langle W^u[j]\rangle_{\mathsf{FL}}, \langle 1\rangle_{\mathsf{FL}})$
12: $\langle c_l\rangle \leftarrow \mathsf{FLLT}(\langle W^l[j]\rangle_{\mathsf{FL}}, \langle 1\rangle_{\mathsf{FL}})$
13: $\langle t\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLChoose}(\langle W^u[j]\rangle_{\mathsf{FL}}, \langle 1\rangle_{\mathsf{FL}}, \langle c_u\rangle)$
14: $\langle W[j]\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLChoose}(\langle W^l[j]\rangle_{\mathsf{FL}}, \langle t\rangle_{\mathsf{FL}}, \langle c_l\rangle)$
15: **end for**
16: **return** $\langle W\rangle_{\mathsf{FL}}$

2.4.4 Weights*

We implement Weights* in Algorithm 4. To allow arbitrary values for ε we avoid costly secure exponentiation for weight computation altogether: Utility u, decomposable w.r.t. u', allows for efficient combination of local weights for D_i, input by the parties, into global weights for D via multiplication (as described in Section 2.3.2).

2.4.5 Running Time Complexity Analysis

We analyse the running time of EM* w.r.t. MPC protocols from Table 2.2 (omitting non-interactive addition/subtraction), and their complexity is given in Table 2.3. The table lists the complexities for MPC protocols typically measured in the number of *rounds* and *interactive operations*, where rounds describes the count of sequential interactive operations, and interactive operations (e.g., reconstruct sharing, multiplications) require each party to send messages to all other parties. We omit integer addition/subtraction as these operations are non-interactive and the parties can perform them locally. Share reconstruction is denoted with Rec. Note that $\mathsf{Choose}(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ is implemented with one multiplication and two additions ($b + (a-b)\cdot c$), and that IntRand uses correlated randomness already exchanged in the offline phase (hence zero interaction and rounds).

We measure the running time of our implementation in Section 2.5.

Theorem 4. EM* *with* $\mathsf{Weights}^{\ln(2)}$ *or* $\mathsf{Weights}^{\ln(2)/2^d}$ *requires* $O(k\lceil\log_k|U|\rceil)$ *MPC protocol calls, with* Weights* *we require* $O(mk\lceil\log_k|U|\rceil)$. *Note that complexity of these MPC protocols is at most* $O(l_v \log l_v + l_x)$ *for bit-lengths* l_v, l_x.

Proof. EM* invokes the weight computation and Select at most $\lceil\log_k|U|\rceil$ times. An invocation of $\mathsf{Weights}^{\ln(2)}$ or $\mathsf{Weights}^{\ln(2)/2^d}$ performs k truncations IntTrunc, $2k$ comparisons IntLT and $2k$ selections IntChoose. Additionally, $\mathsf{Weights}^{\ln(2)/2^d}$ also requires one truncation IntTrunc, modulo IntMod2m, float selection FLChoose and float multiplication FLMul. Weight computation via

Protocol	*Rounds*	*Interactive Operations*
Rec	1	1
IntRand	0	0
IntMod2m	$O(1)$	$O(t)$
IntTrunc	4	$4t+1$
IntLT	4	$4b-2$
Int2FL	$\log v+13$	$\log v(2v-3)-11$
FLAdd	$O(\log v)$	$O(v\log v+x)$
FLMul	$8v+10$	11
FLLT	6	$4v+5x+4\log x+13$

Table 2.3: Complexity of MPC protocols for b-bit integers, t-bit truncation modulus, and floats with v-bit significand and x-bit exponent [ABZS13, CDH10, AKR+20, EKM+14].

Weights* requires $2km$ float multiplications FLMul, $2k$ float comparisons FLLT and $2k$ float selections FLChoose. Each invocation of Select requires $k-1$ float additions FLAdd, only one random draw IntRand, conversion Int2FL and float multiplication FLMul. Also, Select performs at most $\log_2(k)$ comparisons FLLT and share reconstruction steps during binary search. □

2.4.6 Security

We consider the *semi-honest model* introduced by Goldreich [Gol09] where corrupted protocol participants do not deviate from the protocol but gather everything created during the run of the protocol. Our protocol consists of multiple subroutines realized with MPC protocols listed in Table 2.2 (for details and security proof references we refer to [AKR+20]). To analyze the security of the entire protocol we rely on the well-known *composition theorem* [Gol09, Section 7.3.1]. Basically, MPC protocols using an ideal functionality (a subroutine provided by a trusted third party) remain secure if the ideal functionality is replaced with an MPC protocol implementing the same functionality. We implement such ideal functionality with the maliciously secure SCALE-MAMBA framework [AKR+20] (which was faster than its semi-honest fork in a WAN). Our protocol performs multiple subrange selections and each selection round is maliciously secure. Overall, we only provide semi-honest security as malicious adversaries can deviate from inputs provided in previous rounds. We later show how to extend our protocol to malicious adversaries, but first we proof semi-honest security for EM*:

Theorem 5. *Protocol* EM* *realizes* $\mathscr{F}_{\mathsf{EM}^*}$ *in the presence of semi-honest adversaries.*

Proof. To prove semi-honest security we show the existence of a *simulator* Sim according to Goldreich [Gol09] such that the distributions of the protocol transcript EM* is computationally indistinguishable from simulated transcript using $\mathscr{F}_{\mathsf{EM}^*}$ produced in an "ideal world" with a trusted third party. Note that an adversary in the ideal world learns nothing except the protocol inputs and outputs, hence, if he cannot distinguish simulated transcripts (from ideal world) and actual transcripts (in the real world), he learns nothing in our real-world implementation. Next, we formalize the ideal and real-world executions, ideal and real, with notation from Evans et al. [EKR18]: Consider a subset of corrupted parties $C \subset \mathscr{P}$, and let $\texttt{VIEW}_i$ denote the view of party $i \in C$ during the execution of EM* implementing ideal functionality $\mathscr{F}_{\mathsf{EM}^*}$, including all exchanged messages and internal state, and let x_i denote the protocol input of party P_i and $\widehat{\mu}$ the final output of all

parties. The parameters s, k, U are public. Then, $\mathsf{real}_{\mathsf{EM}^*}$, on input security parameter κ, C and all x_i, runs protocol EM^* (where each party P_i behaves honestly using its own input x_i) and outputs $\{\mathtt{VIEW}_i | i \in C\}, \widehat{\mu}$. And $\mathsf{ideal}_{\mathscr{F}_{\mathsf{EM}^*},\mathsf{Sim}}$, with the same inputs, computes $\widehat{\mu} \leftarrow \mathscr{F}_{\mathsf{EM}^*}(x_1, \ldots, x_m)$ and outputs $\mathsf{Sim}(C, \widehat{\mu}, \{x_i \mid i \in C\}), \widehat{\mu}$. Now, simulator Sim produces a transcript for $\mathsf{real}_{\mathsf{EM}^*}$ as follows: As we operate on secret shares, which look random to the parties [EKR18], Sim replaces all secret shares with random values to create $\mathtt{VIEW}_i$. Likewise, the secret-shared output of the weight computations (Algorithm 3 and 4) are replaced with randomness. Sim can simulate Algorithm 2 by recursively splitting U into k subranges, and outputting the subrange containing $\widehat{\mu}$ in each selection step. Finally, Sim outputs a uniform random element from the last subrange (Algorithm 1). Altogether, a semi-honest adversary cannot learn more than the (ideal-world) simulator as this information is sufficient to produce a transcript of our (real-world) protocol. □

For malicious adversaries, we need to ensure consistency between rounds based on Aggarwal et al. [AMP10], who securely compute the (non-DP) median via comparison-based pruning rounds. Informally, we have two consistency constraints: First, valid rank inputs must be monotone within a step. Second, for consistency between steps, valid inputs are contained in the subrange output in the previous step. Formally, let $\{R_1^i, \ldots, R_k^i\}$ denote the set of subranges in the i^{th} step of EM^* and let l_j^i, u_j^i denote the lower resp. upper range endpoint of R_j^i. Then, $\mathrm{rank}_{D_p}(l_1^i) \leq \mathrm{rank}_{D_p}(l_2^i) \leq \cdots \leq \mathrm{rank}_{D_p}(l_k^i) \leq \mathrm{rank}_{D_p}(u_k^i)$ describes monotone input in step i for party p. Consistency between step i and $i+1$, if the j^{th} range was selected, is expressed as $\mathrm{rank}_{D_p}(l_1^{i+1}) = \mathrm{rank}_{D_p}(l_j^i)$ and $\mathrm{rank}_{D_p}(u_k^{i+1}) = \mathrm{rank}_{D_p}(u_j^i)$. In other words, the subrange output in the previous step is used in the current step. Analogously, we can enforce consistency for weights as they are based on rank values.

2.4.7 Scaling to Many Parties

Recall, we distinguish two sets of parties: *Input parties* send shares of their input to *computation parties* which run the secure computation on their behalf. The latter can be a subset of the input parties or non-colluding untrusted servers (e.g., multiple cloud service providers). This scales nicely as the number of computation parties is independent of the number of input parties and can be constant, e.g., 3. In our evaluation (Section 2.5) $m \in \{3, 6, 10\}$ computation parties perform the computation for 10^6 input parties, each holding a single datum. Addition suffices for $\mathsf{Weights}^{\ln(2)}$ and $\mathsf{Weights}^{\ln(2)/2^d}$ to combine local rank values into a global rank. Addition is essentially "free" as it requires no interaction between the computation parties. For $\mathsf{Weights}^*$ we require multiplication to combine the local weights, which requires interaction during the preprocessing step. However, $\log n$ rounds suffice to combine the inputs by building a tree of pairwise multiplications with 2^i multiplications at level i [ABZS13].

2.5 Evaluation

Our implementation is realized with the SCALE-MAMBA framework [AKR+20] using Shamir secret sharing with a 128-bit modulus and honest majority. Next, we evaluate the running time, communication, privacy budget, and accuracy of our solution.

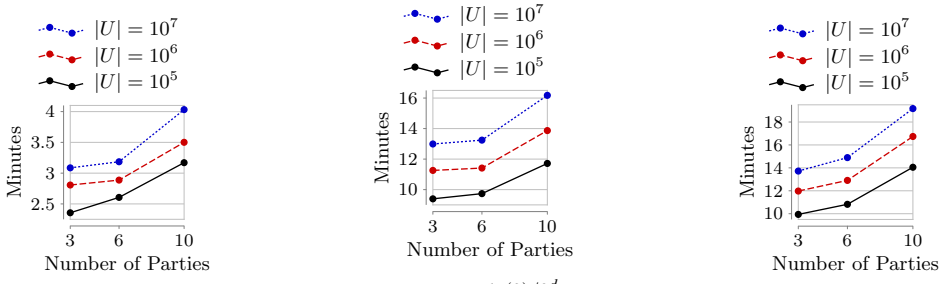
Protocol	$\lvert U\rvert$	*Running time*
Eigner et al. [EKM+14]	5	42.3 s
EM* & Weights$^{\ln(2)}$	10^5	11.3 s (7.7 s)
	10^6	13.5 s (9.2 s)
	10^7	15.4 s (10.7 s)
EM* & Weights$^{\ln(2)/2^d}$, $d=2$	10^5	33.7 s (23.6 s)
	10^6	39.8 s (27.8 s)
	10^7	46.8 s (31.4 s)
EM* & Weights*	10^5	64.3 s (41.6 s)
	10^6	77.3 s (52.4 s)
	10^7	91.8 s (61.1 s)

Table 2.4: Running times for 3 parties in a 1 Gbits/s LAN for this work and Eigner et al. [EKM+14]. We report the average of 20 runs on t2.medium instances with 4 vCPUs, 2 GB RAM (and r4.2xlarge instances with 8 vCPUs, 61 GB RAM). Eigner et al. [EKM+14] evaluated on a 3.20 GHz, 16 GB RAM machine.

2.5.1 Running Times

We evaluated on t2.medium AWS instances with 2GB RAM, 4 vCPUs [Ama20] and the Open Payments dataset from the Centers for Medicare & Medicaid Services (CMS) [fMMS17]. Our evaluation uses 10^6 records from the Open Payments dataset, however, our approach scales to any dataset size as we consider universe subranges. We used the maximum number of selection steps, i.e., $s = \lceil \log_k |U| \rceil$, with $k = 10$ ranges per step. We evaluated the average running time of 20 runs of the *entire* protocol EM*, i.e., offline as well as online phase, in a LAN and a WAN.

LAN: We measured our running time for 3 parties in a LAN with 1 Gbits/s bandwidth to compare it to Eigner et al. [EKM+14] who only report LAN running times. We support universe sizes of more than 5 orders of magnitude larger with comparable running times: They compute weights per elements and require around 42 seconds for $|U| = 5$, whereas our protocol EM* using Weights$^{\ln(2)}$/ Weights$^{\ln(2)/2^d}$/ Weights* runs in approx. 11 / 33 / 64 seconds for $|U| = 10^5$. For detailed measurements see Table 2.4. Eigner et al. [EKM+14] evaluated their protocol with a sum utility function on a 3.20 GHz, 16 GB RAM machine. They are linear in the size of the universe and compute weights for a very small universe of only 5 elements. We, on the other hand, are sublinear in the size of the universe as we compute weights per subrange and use efficient alternatives to costly secure exponentiation. We evaluated universe sizes at least 5 order of magnitudes larger than [EKM+14] with comparable running times: Our running time for Weights$^{\ln(2)}$, Weights$^{\ln(2)/2^d}$ is below [EKM+14] on rather modest t2.medium instances (4 vCPUs, 2 GB RAM) for universe size $|U| = 10^5$. Even if we also consider weights per element (i.e., subrange size 1) for any decomposable utility function our protocols compute at least 6 times more weights per second on t2.medium instances. (E.g., for $k = 10, |U| = 10^5$ and Weights* we compute 50 weights in 64.3 seconds, i.e., 0.78 weights per second, compared to 0.12 for [EKM+14].) We also evaluated our protocol on r4.2xlarge instances (8 vCPUs, 61 GB RAM), see seconds in parenthesis in Table 2.4. In a LAN the running time compared to t2.medium instances is reduced by at least 30%, however, in a WAN setting the latency plays a more important role than powerful hardware and the running times are much closer. Thus, we only present running times for t2.medium instances in a WAN in Section 2.5.



(a) $\mathsf{Weights}^{\ln(2)}$ (b) $\mathsf{Weights}^{\ln(2)/2^d}$, $d = 2$ (c) $\mathsf{Weights}^*$

Figure 2.3: Average running time of EM^* – with weight computation subroutines $\mathsf{Weights}^{\ln(2)}$, $\mathsf{Weights}^{\ln(2)/2^d}$, or $\mathsf{Weights}^*$– for 20 runs on t2.medium instances in Ohio and Frankfurt (100 ms delay, 100 Mbits/s bandwidth).

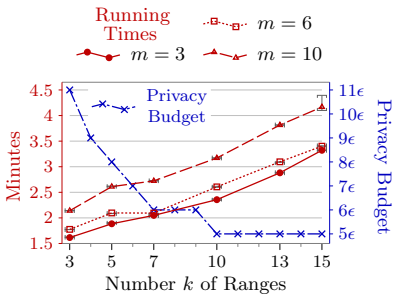
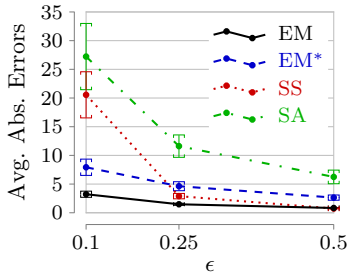


Figure 2.4: Privacy vs. running time trade-off: For increasing number k of subranges the running time (left axis) increases whereas the consumed privacy budget (right axis) decreases. (Illustrated for EM^* with $\mathsf{Weights}^{\ln(2)}$ and $|U| = 10^5$).

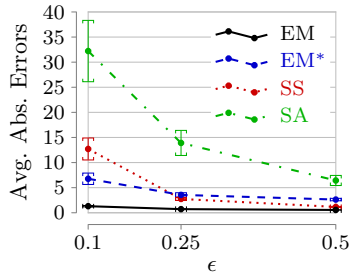
WAN: We consider m computation parties, which already received and combined secret-shared inputs from 10^6 users (Section 2.4.7), and report the average running time of our protocol. We split the m parties into two regions, Ohio (us-east-2) and Frankfurt (eu-central-1), and measured an inter-region round time trip (RTT) of approx. 100 ms with 100 Mbits/s bandwidth. We evaluated all weight computation subroutines in Figure 2.3 for $m \in \{3, 6, 10\}$ computation parties and $|U| \in \{10^5, 10^6, 10^7\}$. The results are very stable, as the 95% confidence intervals deviate by less than 0.5% on average. $\mathsf{Weights}^{\ln(2)}$ (Figure 2.3a) is the fastest with running times around 3 minutes for 3 parties, whereas $\mathsf{Weights}^{\ln(2)/2^d}$ (Figure 2.3b) and $\mathsf{Weights}^*$ (Figure 2.3c) require around 13 and 14 minutes respectively. However, we consider large universe sizes (billions of elements) in a real-world network with large latency. The choice of weight computation enables a trade-off between faster running times, i.e., $\mathsf{Weights}^{\ln(2)}$ with fixed ε, and smaller privacy loss ε, i.e, $\mathsf{Weights}^*$, with $\mathsf{Weights}^{\ln(2)/2^d}$ positioned in the middle (faster running time than $\mathsf{Weights}^*$ with smaller ε compared to $\mathsf{Weights}^{\ln(2)}$). The number k of subranges allow a similar trade-off, as discussed next.

2.5.2 Privacy Budget vs. Running Time

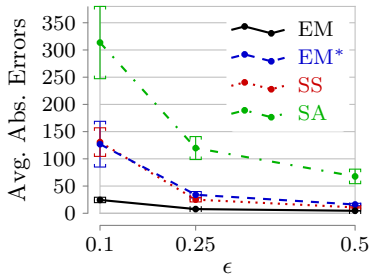
The *privacy budget* is the sum of privacy parameters consumed per step, i.e., the overall privacy loss. Figure 2.4 shows how the privacy budget and the running time are affected by the number k of subranges. Larger k leads to larger running times, as the number of costly secure computations de-



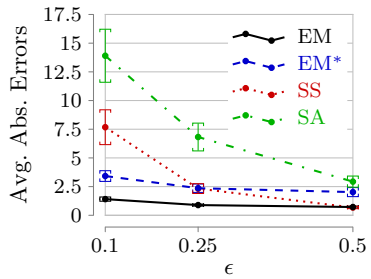
(a) Credit card data [ULB18], first 10^5 payment records in Cents.



(b) Walmart supply chain data [Kag18], 175k shipment weights as integers.



(c) California public salaries [Soo18], 71k records, state department's total wages.



(d) Normal distribution with $\sigma = 3$, 10^5 samples (as integers with scaling factor 1000).

Figure 2.5: Comparing exponential mechanism (EM) as baseline, this work (EM*), smooth sensitivity (SS) [NRS07], sample-and-aggregate (SA) [PL15] on different data, 100 averaged runs.

pends on the number of ranges times the number of selection steps ($k \cdot \lceil \log_k |U| \rceil$), which increases proportionally to k. However, smaller values for k require more selection steps ($\lceil \log_k |U| \rceil$), which lead to an increase in the privacy budget. Overall, for $k = 10$ subranges, as used in our evaluation, the consumed privacy budget is small with an acceptable running time.

2.5.3 Accuracy Comparison to Related Work

EM* performs multiple selection steps s, each consume a portion ε_i of the overall privacy budget $\varepsilon = \sum_{i=1}^{s} \varepsilon_i$. How to optimally split ε (optimal composition) is #P-complete [MV16]. Thus, we use the following heuristic to divide ε among the selection steps: Initial steps cover exponentially larger subranges, and require exponentially less of the privacy budget. After a while an equal split is more advantageous, as the subranges become smaller and contain fewer elements. Altogether, we use $\varepsilon_i = \varepsilon / 2^{s-i+1}$ if $i < \lfloor s/2 \rfloor - 1$ and $\varepsilon_i = \varepsilon' / (s - \lfloor s/2 \rfloor)$ else, where ε' is the remaining privacy budget. We used $s = \lceil \log_k |U| \rceil - 1$ for our accuracy evaluation. We found in our experiments that performing one selection step less increases accuracy, as the privacy budget can be better divided among the other remaining steps and the last subrange is already small enough (at most k elements).

Related work computing DP median in the central model shows a strong data dependence which makes straightforward comparison difficult. Therefore, we empirically evaluated the different approaches closest to ours, i.e., supporting more than 2 parties, on real-world datasets [Kag18, ULB18, Soo18] as well as the normal distribution in Figure 2.5[8] for 100 averaged runs with 95%-confidence intervals. Low ε (as evaluated) is desirable as it provides more privacy or

[8] "Small" data is the most challenging regime for DP [NRVW20, BEM$^+$17], thus, we use small datasets to better illustrate the accuracy differences.

Protocol	$\lvert U\rvert$	*Communication* $m=3$	$m=6$	$m=10$
EM^* & $\mathsf{Weights}^{\ln(2)}$	10^5	178 MB	402 MB	1.41 GB
	10^6	202 MB	448 MB	1.54 GB
	10^7	222 MB	497 MB	1.75 GB
EM^* & $\mathsf{Weights}^{\ln(2)/2^d}$, d=2	10^5	634 MB	1.38 GB	4.73 GB
	10^6	748 MB	1.63 GB	5.58 GB
	10^7	866 MB	1.88 GB	6.39 GB
EM^* & $\mathsf{Weights}^*$	10^5	664 MB	1.56 GB	5.59 GB
	10^6	785 MB	1.83 GB	6.57 GB
	10^7	907 MB	2.11 GB	7.59 GB

Table 2.5: Communication cost (WAN with 100 Mbits/s and 100 ms latency): Data sent per party, average of 20 runs for $m \in \{3,6,10\}$ parties and $|U| \in \{10^5, 10^6, 10^7\}$.

allows the remaining privacy budget to be spend on additional queries. The evaluation for smooth sensitivity [NRS07] and exponential mechanism per element assume a trusted party with full access to the dataset, whereas our approach and [PL15] use MPC instead of a trusted party. Nissim et al. [NRS07] (SS in Figure 2.5) compute instance-specific additive noise, requiring full data access, and achieve good accuracy, however, the exponential mechanism can provide better accuracy for low ε. Pettai & Laud [PL15] (SA in Figure 2.5) securely compute the noisy average of the 100 values closest to the median within a clipping range. Recall, the median is the 0.5^{th}-percentile. To minimize the error from clipping range $[c_l, c_u]$, we choose $c_l = 0.49^{\text{th}}$-percentile, $c_u = 0.51^{\text{th}}$-percentile, i.e., we presume to already know a tight range for the actual median. Nonetheless, in our experiments the absolute error of SA is the largest. Overall, no solution is optimal for all ε and datasets. However, the exponential mechanism EM, and our protocol EM^*, provide the best accuracy for low ε, i.e., high privacy, compared to additive noise approaches [NRS07, PL15].

2.5.4 Communication

The communication for the maximum number of steps ($\lceil \log_{10} |U| \rceil$) in a WAN (100 Mbits/s with 100 ms latency) is detailed in Table 2.5. For 3 parties and one billion universe elements, the communication for $\mathsf{Weights}^{\ln(2)}$ is 222 MB per party, for $\mathsf{Weights}^{\ln(2)/2^d}$ it is 866 MB, and $\mathsf{Weights}^*$ requires 907 MB.

We stress that this communication is required for *malicious security* in each round as provided by the SCALE-MAMBA implementation. MP-SPDZ [Kel20], a fork of SCALE-MAMBA's predecessor SPDZ2, also provides semi-honest security. MP-SPDZ with semi-honest security requires much less communication, e.g., only around 25 MB for 3 parties, $|U| = 10^5$, and $\mathsf{Weights}^*$. However, the running time in a WAN was some minutes slower compared to SCALE-MAMBA in our tests (presumably due to SCALE-MAMBA's batched communication rounds and integrated online and offline phases, where parallel threads create offline data "just-in-time" [AS19, AKR+20]). Thus, regarding our protocol, one can choose efficiency w.r.t. communication (MP-SPDZ) or running time (SCALE-MAMBA).

2.5.5 Cost of Malicious Security

To achieve malicious security, via consistency checks as detailed in Section 2.4.6, we require additional running time and communication. For the maximum number of steps with one billion universe elements in a WAN (100 Mbits/s with 100 ms latency) EM^* with $\mathsf{Weights}^*$ additionally needs around 10/10/12 minutes and 0.65/1.4/5 GB for 3/6/10 parties. EM^* with $\mathsf{Weights}^{\ln(2)}$ or $\mathsf{Weights}^{\ln(2)/2^d}$ ($d = 2$) additionally requires around 1.3/1.5/2 minutes and 115/260/825 MB for 3/6/10 parties.

2.6 Related Work

Next, we describe related work for secure computation of the exponential mechanism, DP median and decomposability.

Secure Exponential Mechanism: Alhadidi et al. [AMFD12] present a secure 2-party protocol for the exponential mechanism for max utility functions. It uses garbled circuits and oblivious polynomial evaluation to compute Taylor series for the exponential function. Our work is more general as we support more parties and a broader class of utility functions, including max utility functions. Eigner et al. [EKM+14] present a carefully designed secure exponential mechanism in the multi-party setting. Their work is more general, supporting arbitrary utility functions and malicious parties, but they are linear in the size of the universe, and securely compute the exponential function. We provide a sublinear solution without costly secure exponentiation, supporting at least 5 orders of magnitude more elements. Other work also securely compute the DP median with the exponential mechanism [BK20b] – see Deliverable D5.4 [PFOR21] for details. They optimize their protocol for the 2-party setting, compute the utility over (sorted) data, and provide DP for small data (sublinear in the size of the universe). They initially prune large datasets via [AMP10] (who securely compute the exact median), requiring a relaxation of DP [HMFS17], to achieve running time sublinear in the universe size. We consider the multi-party setting and provide pure differential privacy.

DP Median: Pettai and Laud [PL15] securely compute DP statistics, including the DP median, via sample-and-aggregate [NRS07]. Their implementation is based on secret sharing in a 3-party setting. Pettai and Laud [PL15] compute the DP median as noisy average of 100 values closest to the median within a clipping range, which limits accuracy, especially, if the data contains outliers or large gaps (see Section 2.5.3). Dwork and Lei [DL09] consider robust privacy-preserving statistics with a trusted third party where data samples are known to be drawn (independent and identically distributed) from a distribution. They present the first DP median algorithm that does not require bounds for the data but aborts if the data are not from a "nice" distribution with small sensitivity. Their DP median algorithm first estimates scale s via DP interquartile range and the noise magnitude $sn^{-1/3}$ can be large. Nissim et al. [NRS07] present smooth sensitivity, which analyzes the data to provide instance-specific noise. For the DP median, the exponential mechanism provides better accuracy for low ε and can be efficiently computed, whereas computation of smooth sensitivity requires full data access in clear or the error increases (see Section 2.2.1). PINQ, a DP query framework developed by McSherry [McS09], also computes the DP median via the exponential mechanism, however, they rely on a trusted third party with access to the data in clear. Cryptε [CWH+20] employs two non-colluding untrusted servers and cryptographic primitives to compute noisy histograms (Laplace mechanism) for SQL queries (e.g., count, distinct count) in the central model, which can be extended to compute the median. However, we

show that the exponential mechanism is more accurate for the median with low ε. Also, Cryptε has a running time linear in the data size, whereas our work is independent of the data size. Smith et al. [STU17] and Gaboardi et al. [GSX18] consider the restrictive non-interactive local model, where at most one message is sent from client to server, and achieve optimal local model error. However, local DP requires more samples to achieve the same accuracy as central DP. (No non-interactive LDP-protocol [STU17, GSX18] can achieve asymptotically better sample complexity than $O(\varepsilon^{-2}\alpha^{-2})$ for error α [DJW13].) We, on the other hand, are interested in high accuracy, as in the central model, even for small sample sizes. We provide an empirical comparison to related work in Section 2.5.3.

Decomposability: MapReduce is a programming paradigm for distributed data aggregation where a mapper produces intermediary results (e.g., partial sums) that a reducer combines into a result (e.g., total sum). Airavat [RSK+10] provide a Hadoop-based MapReduce programming platform for DP statistics based on additive noise (Laplace mechanism) with an untrusted mapper but trusted reducer. We consider decomposable utility functions for the exponential mechanism without any trusted parties. The secure exponential mechanisms [AMFD12,EKM+14] use decomposable utility functions (max and counts), but do not classify nor provide optimizations for such functions. Blocki et al. [BDB16] minimize cummulative error for DP password frequency lists employing (decomposability of) frequencies for their dynamic programming, which has access to all the data in the clear. We use decomposable aggregate functions to efficiently and securely combine inputs.

2.7 Summary

We presented a novel alternative for differentially private median computation with high accuracy (even for small number of users), without a trusted party, that is efficiently computable (practical running time) and scaleable (sublinear in the size of the universe). Our semi-honest multi-party protocol implements the exponential mechanism for decomposable aggregate functions (e.g., rank-based statistics) as used in MapRedue-style algorithms, and can be extended to malicious parties. For the median, the exponential mechanism provides the best utility vs. privacy trade-off for low ε in our evaluations of related work in the central model. We optimize our protocol for decomposable functions (allowing efficient MPC on distributed data), and use efficient alternatives to exponentiations for floating-point numbers. We implemented our protocol in the SCALE-MAMBA framework [AKR+20], and evaluated it for 1 million users using 3 semi-honest computation parties achieving a running time of seconds in a LAN, and 3 minutes in a WAN (100 ms latency, 100 Mbits/s bandwidth).

3. Conclusions

This document reported on the complete techniques for sanitization and computation. The focus of the discussed techniques is a distributed scenario where multiple parties collaboratively perform sanitization. The tools and techniques detailed in this document satisfy well-known and formal privacy notions; namely, k-anonymity, l-diversity, and differential privacy.

Chapter 1 detailed techniques to efficiently perform anonymization satisfying k-anonymity and l-diversity for large scale data collections. The chapter described the advancements and progress for scalable anonymization developed during MOSAICrOWN. The focus was on an extension of Mondrian. Mondrian efficiently provides k-anonymity but was originally developed for a centralized setting. The presented approach extended Mondrian to handle distributed computations by leveraging a set of workers performing parallelized processing. The approach presented a dynamic partitioning technique that does not require knowledge of the entire data set and also supports various generalizations strategies to satisfy the privacy notions. Experimental evaluations show that the approach scales nicely with the number of workers and is well suited for distributed anonymization.

Chapter 2 presented collaborative, cryptographic protocols to securely compute differentially private statistics with data distributed across multiple parties. In more detail, the solution supports statistics based on decomposable aggregate functions (e.g., based on counts, ranks) as found in MapReduce-style frameworks. A secure multi-party computation implemented with the SCALE-MAMBA framework is detailed, that efficiently computes the exponential mechanism for problems whose score can be expressed as decomposable aggregate functions. The exponential mechanism scores each possible output element from the data domain, and elements with higher utility scores are selected with higher probability. Efficiency is ensured in two ways. Firstly, by partitioning the large domain in small subranges and iteratively, privately selecting increasingly smaller subranges until one contains the statistical value of interest. Secondly, by providing novel approaches to securely compute exponentiations. Altogether, the efficiency is superior to related work that securely computes the exponential mechanism.

Bibliography

[Abo18] J. M. Abowd. The U.S. Census Bureau Adopts Differential Privacy. In *Proc. of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[ABZS13] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In *Network and Distributed Systems Security Symposium*, NDSS, 2013.

[AKR+20] A. Aly, M. Keller, D. Rotaru, P. Scholl, N. P. Smart, and T. Wood. SCALE-MAMBA Documentation. `https://homes.esat.kuleuven.be/~nsmart/SCALE/`, 2020.

[AKS21] F. Ashkouti, K. Khamforoosh, and A. Sheikhahmadi. DI-Mondrian: Distributed improved Mondrian for satisfaction of the ℓ-diversity privacy model using Apache Spark. *Information Sciences*, 546:1–24, 2021.

[Ama20] Amazon.com. Amazon Web Services. `https://aws.amazon.com/ec2/pricing/on-demand/`, 2020.

[AMFD12] D. Alhadidi, N. Mohammed, B. CM Fung, and M. Debbabi. Secure distributed framework for achieving ε-differential privacy. In *International Symposium on Privacy Enhancing Technologies Symposium*, PETS, 2012.

[AMP10] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the median (and other elements of specified ranks). *Journal of Cryptology*, 2010.

[App16] Apple. Wwdc 2016: Engineering privacy for your users, 2016. `https://developer.apple.com/videos/play/wwdc2016/709/`.

[AS19] A. Aly and N. P Smart. Benchmarking privacy preserving scientific operations. In *International Conference on Applied Cryptography and Network Security*, ACNS, 2019.

[BA05] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proc. of ICDE 2005*, Tokoyo, Japan, April 2005.

[BDB16] J. Blocki, A. Datta, and J. Bonneau. Differentially private password frequency lists. In *Network and Distributed Systems Security Symposium*, NDSS, 2016.

[BDOZ11] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, 2011.

[Bea91] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, CRYPTO, 1991.

[BEM+17] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proc. of the Symposium on Operating Systems Principles*, SOSP, 2017.

[BK20a] J. Böhler and F. Kerschbaum. Secure multi-party computation of differentially private median. In *USENIX Security Symposium*, USENIXSec, 2020.

[BK20b] J. Böhler and F. Kerschbaum. Secure sublinear time differentially private median computation. In *Network and Distributed Systems Security Symposium*, NDSS, 2020.

[BK21] J. Böhler and F. Kerschbaum. Secure Multi-party Computation of Differentially Private Heavy Hitters. In *Computer and Communications Security*, CCS, 2021.

[BST14] R. Bassily, A. Smith, and A. Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2014.

[BWAA18] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.

[CDF+10] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM TISSEC*, 13(3):22:1–22:33, July 2010.

[CDFS07] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-Anonymity. In T. Yu and S. Jajodia, editors, *Secure Data Management in Decentralized Systems*. Springer-Verlag, 2007.

[CDFS09] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. Theory of privacy and anonymity. In M. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook (2nd edition)*. CRC Press, 2009.

[CDH10] O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, SCN, 2010.

[CWH+20] A. R. Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha. Cryptε: Crypto-assisted differential privacy on untrusted servers. In *Proc. of the annual ACM SIGMOD International Conference on Management of data*, SIGMOD, 2020.

[CWR14] A. Chakravorty, T. W. Wlodarczyk, and C. Rong. A scalable k-anonymization solution for preserving privacy in an aging-in-place welfare intercloud. In *Proc. of IC2E 2014*, Boston, MA, USA, March 2014.

[DFF+21a] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Artifact: Scalable distributed data anonymization. In *Proc. of IEEE PerCom 2021*, Kassel, Germany, March 2021.

[DFF+21b] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Scalable distributed data anonymization. In *Proc. of IEEE PerCom 2021*, Kassel, Germany, March 2021.

[DFJ+15] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Loose associations to increase utility in data publishing. *JCS*, 23(1):59–88, 2015.

[DFK+06] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, TCC, 2006.

[DFLS11] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Anonymization of statistical data. *IT - Information Technology*, 53(1):18–25, January 2011.

[DFLS12] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Data privacy: Definitions and techniques. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 20(6):793–817, December 2012.

[DG08] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.

[DJW13] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2013.

[DKM+06] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, 2006.

[DKY17] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*, NeurIPS, 2017.

[DL09] C. Dwork and J. Lei. Differential privacy and robust statistics. In *Proc. of the annual ACM symposium on Theory of Computing*, STOC, 2009.

[DMNS06] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, TCC, 2006.

[DPSZ12] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual International Cryptology Conference*, CRYPTO, 2012.

[DR14] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[Dwo06] C. Dwork. Differential privacy. In *Proc. of ICALP 2006*, Venice, Italy, 2006.

[DYL+13] X. Ding, Q. Yu, J. Li, J. Liu, and H. Jin. Distributed anonymization for multiple data providers in a cloud system. In *Proc. of DASFAA 2013*, Wu Han, China, April 2013.

[EKM+14] F. Eigner, A. Kate, M. Maffei, F. Pampaloni, and I. Pryvalov. Differentially private data aggregation with optimal utility. In *Proc. of the Annual Computer Security Applications Conference*, ACSAC, 2014.

[EKR18] D. Evans, V. Kolesnikov, and M. Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2018.

[EPK14] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proc. of the annual ACM conference on computer and communications security*, CCS, 2014.

[fMMS17] Centers for Medicare & Medicaid Services. Complete 2017 program year open payments dataset, 2017. `https://www.cms.gov/OpenPayments/Explore-the-Data/Dataset-Downloads.html`.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of the annual ACM symposium on Theory of Computing*, STOC, 1987.

[Gol09] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2009.

[GSX18] M. Gaboardi, A. Smith, and J. Xu. Empirical risk minimization in the non-interactive local model of differential privacy. In *Proc. of NIPS 2018*, Montréal, Canada, December 2018.

[GX17] S. Goryczka and L. Xiong. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[HMFS17] X. He, A. Machanavajjhala, C. Flynn, and D. Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proc. of the annual ACM conference on Computer and Communications Security*, CCS, 2017.

[JX09] P. Jurczyk and L. Xiong. Distributed anonymization: Achieving privacy for both data subjects and data providers. In *Proc. of DBSec 2009*, Montreal, Canada, July 2009.

[Kag18] Kaggle.com. Walmart supply chain: Import and shipment. `https://www.kaggle.com/sunilp/walmart-supply-chain-data/data`, 2018. Retrieved: October, 2019.

[Kam15] L. Kamm. *Privacy-preserving statistical analysis using secure multi-party computation*. PhD thesis, PhD thesis, University of Tartu, 2015.

[Kel20] M. Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proc. of the annual ACM conference on Computer and Communications Security*, CCS, 2020.

[KLN+11] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 2011.

[KPEK14] F. Kohlmayer, F. Prasser, C. Eckert, and K. A. Kuhn. A flexible approach to distributed data anonymization. *Journal of Biomedical Informatics*, 50:62–76, 2014.

[KPR18] M. Keller, V. Pastro, and D. Rotaru. Overdrive: making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, 2018.

[KRSW18] M. Keller, D. Rotaru, N. P. Smart, and T. Wood. Reducing communication channels in mpc. In *International Conference on Security and Cryptography for Networks*, SCN, 2018.

[LDR05] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proc. of SIGMOD 2005*, June 2005.

[LDR06] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proc. of ICDE 2006*, Atlanta, GA, USA, April 2006.

[LLSY16] N. Li, M. Lyu, D. Su, and W. Yang. Differential privacy: From theory to practice. *Synthesis Lectures on Information Security, Privacy, & Trust*, 2016.

[LLV07] N. Li, T. Li, and S. Venkatasubramanian. t-Closeness: Privacy beyond k-anonymity and l-diversity. In *Proc. of ICDE 2007*, Istanbul, Turkey, April 2007.

[LLV10] N. Li, T. Li, and S. Venkatasubramanian. Closeness: A new privacy measure for data publishing. *IEEE TKDE*, 22(7):943–956, 2010.

[LP09] Y. Lindell and B. Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology*, 2009.

[McS09] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proc. of the annual ACM SIGMOD International Conference on Management of data*, SIGMOD, 2009.

[MGK06] A. Machanavajjhala, J. Gehrke, and D. Kifer. ℓ-diversity: Privacy beyond k-anonymity. In *Proc. of ICDE 2006*, Atlanta, GA, USA, April 2006.

[MKGV07] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. *ACM TKDD*, 1(1):3:1–3:52, 2007.

[MPRV09] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *Annual International Cryptology Conference*, CRYPTO, 2009.

[MT07] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2007.

[MV16] J. Murtagh and S. Vadhan. The complexity of computing the optimal composition of differential privacy. In *Theory of Cryptography Conference*, TCC, 2016.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Annual International Cryptology Conference*, CRYPTO, 2012.

[NRS07] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proc. of the annual ACM symposium on Theory of Computing*, STOC, 2007.

[NRVW20] S. Neel, A. Roth, G. Vietri, and Z. S. Wu. Oracle Efficient Private Non-Convex Optimization. In *Proc. of the International Conference on Machine Learning*, PMLR, 2020. `https://proceedings.icml.cc/static/paper_files/icml/2020/354-Paper.pdf`.

[PFOR21] S. Paraboschi, D. Facchinetti, G. Oldani, and M. Rossi. D5.4 – Final versions of tools for data sanitisation and computation. Technical report, MOSAICrOWN, 2021.

[PL15] M. Pettai and P. Laud. Combining differential privacy and secure multiparty computation. In *Proc. of the Annual Computer Security Applications Conference*, ACSAC, 2015.

[RN10] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proc. of the annual ACM SIGMOD International Conference on Management of data*, SIGMOD, 2010.

[RSK+10] I. Roy, S. TV Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2010.

[SA17] U. Sopaoglu and O. Abul. A top-down *k*-anonymization implementation for Apache Spark. In *Proc. of IEEE Big Data 2017*, Boston, MA, USA, December 2017.

[Sam01] P. Samarati. Protecting respondents' identities in microdata release. *IEEE TKDE*, 13(6):1010–1027, November/December 2001.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 1979.

[Soo18] G. Sood. California Public Salaries Data, 2018. https://doi.org/10.7910/DVN/KA3TS8.

[STU17] A. Smith, A. Thakurta, and J. Upadhyay. Is interaction necessary for distributed private learning? In *IEEE Symposium on Security and Privacy*, SP, 2017.

[Tea17] Apple's Differential Privacy Team. Learning with privacy at scale, 2017. `https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html`.

[TG12] T. Tassa and E. Gudes. Secure distributed computation of anonymized views of shared databases. *ACM TODS*, 37(2):1–43, 2012.

[TKZ16] H. Takabi, S. Koppikar, and S. T. Zargar. Differentially private distributed data analysis. In *IEEE International Conference on Collaboration and Internet Computing*, CIC, 2016.

[ULB18] Machine Learning Group ULB. Credit card fraud detection, 2018. `https://www.kaggle.com/mlg-ulb/creditcardfraud/data`.

[U.S19] U.S. Bureau of the Census. Public Use Microdata Sample. Individual dataset of all US. 1-Year version of ACS 2019. `https://www2.census.gov/programs-surveys/acs/data/pums/2019/1-Year`, 2019.

[XT06] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *Proc. of VLDB 2006*, Seoul, South Korea, September 2006.

[XWP+06] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A.W.-C. Fu. Utility-based anonymization for privacy preservation with less information loss. *ACM SIGKDD Explorations Newsletter*, 8(2):21–30, 2006.

[Yao86] A. C.-C. Yao. How to generate and exchange secrets. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 1986.

[ZLD+16] X. Zhang, C. Leckie, W. Dou, J. Chen, R. Kotagiri, and Z. Salcic. Scalable local-recoding anonymization using locality sensitive hashing for big data privacy preservation. In *Proc. of CIKM 2016*, Indianapolis, IN, USA, October 2016.

[ZYLC13] X. Zhang, L. T Yang, C. Liu, and J. Chen. A scalable two-phase top-down specialization approach for data anonymization using MapReduce on cloud. *IEEE TPDS*, 25(2):363–373, 2013.